

UCLA

UCLA Electronic Theses and Dissertations

Title

Applications of Probabilistic Reasoning in Trustworthy AI: From Handling Missing Data to Explainability

Permalink

<https://escholarship.org/uc/item/59x011kp>

Author

Khosravi, Pasha

Publication Date

2023

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA

Los Angeles

Applications of Probabilistic Reasoning in Trustworthy AI:  
From Handling Missing Data to Explainability

A dissertation submitted in partial satisfaction  
of the requirements for the degree  
Doctor of Philosophy in Computer Science

by

Pasha Khosravi

2023

© Copyright by  
Pasha Khosravi  
2023

## ABSTRACT OF THE DISSERTATION

Applications of Probabilistic Reasoning in Trustworthy AI:  
From Handling Missing Data to Explainability

by

Pasha Khosravi

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2023

Professor Guy Van den Broeck, Chair

Machine Learning models are becoming increasingly ubiquitous in our daily lives. Despite their advancements, common challenges such as missing data, outliers, and complex behaviors still hinder their usability and trustworthiness. Handling uncertainty is a fundamental aspect of modern machine learning which can potentially help with these challenges. Specifically, in this thesis, we highlight the strengths of tractable probabilistic models, such as probabilistic circuits, in mitigating these issues.

First, we introduce "Expected Prediction" (EXP) as a new type of probabilistic query and examine its computational complexity for a few families of models. We then demonstrate how computing EXP can help address the additional uncertainty arising from missing data in a principled manner. Next, we illustrate how EXP can be used to generate sufficient explanations for classifier decisions while offering probabilistic guarantees. Finally, we leverage the supermodularity of marginal queries and continuous relaxations like multi-linear extension to scale the detection of the root causes of outliers in high-dimensional settings.

The dissertation of Pasha Khosravi is approved.

Quanquan Gu

Bolei Zhou

Aditya Grover

Guy Van den Broeck, Committee Chair

University of California, Los Angeles

2023

To My Family

## TABLE OF CONTENTS

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Reasoning about Logistic Regression with Missing Features</b>	<b>4</b>
2.1	Introduction	4
2.2	The Expected Prediction Task	6
2.3	Joint Distributions as Classifiers	9
2.4	Naive Conformant Learning	13
2.5	Empirical Evaluation	16
2.6	Case Study: Sufficient Explanations	19
2.7	Related Work	21
<b>3</b>	<b>On Tractable Computation of Expected Predictions</b>	<b>23</b>
3.1	Introduction	23
3.2	Expectations and higher order moments of discriminative models	25
3.3	Generative and discriminative circuits	26
3.4	Computing expectations and moments for circuit pairs	30
3.4.1	EC <sub>2</sub> : Expectations of regression circuits	30
3.4.2	MC <sub>2</sub> : Moments of regression circuits	32
3.4.3	Approximating expectations of classifiers	34
3.5	Expected prediction in action	35
3.5.1	Reasoning with missing values: an application	36
3.5.2	Reasoning about predictive models for exploratory data analysis	38

3.6	Related Work . . . . .	40
<b>4</b>	<b>Handling Missing Data in Decision Trees: A Probabilistic Approach . . . . .</b>	<b>42</b>
4.1	Introduction . . . . .	42
4.2	Background . . . . .	44
4.3	Expected Predictions of Decision Trees . . . . .	46
4.4	Expected Parameter Learning of Trees . . . . .	48
4.5	Experiments . . . . .	50
<b>5</b>	<b>Probabilistic Sufficient Explanations . . . . .</b>	<b>52</b>
5.1	Introduction . . . . .	52
5.2	Background and Related Work . . . . .	54
5.3	Motivation and Problem Statement . . . . .	56
5.4	Probabilistic Notions of Sufficiency . . . . .	57
5.4.1	Probabilistic Reasoning Tools . . . . .	57
5.4.2	Connections between SDP and EP . . . . .	59
5.5	Probabilistic Sufficient Explanations . . . . .	61
5.6	Experiments . . . . .	64
5.6.1	Comparison with Anchors . . . . .	64
5.6.2	Comparison with Logical Explanations . . . . .	67
5.6.3	Tradeoffs between Sufficiency and Simplicity . . . . .	68
<b>6</b>	<b>Why Is This an Outlier? Explaining Outliers by Submodular Optimization of Marginal Distributions. . . . .</b>	<b>70</b>
6.1	Introduction . . . . .	70



6.2	Outlier Feature Detection . . . . .	72
6.3	Submodular Optimization for Efficient Outlier Feature Detection . . . . .	74
6.3.1	Supermodularity and Monotonicity of Marginal Probabilities . . . . .	76
6.3.2	Other greedy approximations (without guarantees) . . . . .	77
6.4	Probabilistic models for tractable marginals of arbitrary subsets . . . . .	78
6.5	Continuous Relaxation of Search for Outlier Features . . . . .	80
6.6	Experiments . . . . .	83
6.7	Related Work . . . . .	91
<b>Appendix A Reasoning about Logistic Regression with Missing Features . . . . .</b>		<b>92</b>
A.1	Proofs . . . . .	92
A.1.1	Proof of Theorem 1 . . . . .	92
A.1.2	Proof of Lemma 2 . . . . .	93
A.1.3	Proof of Lemma 3 . . . . .	94
A.2	Beyond Binary Classification: Multiclass . . . . .	94
<b>Appendix B On Tractable Computation of Expected Predictions . . . . .</b>		<b>97</b>
B.1	Proofs . . . . .	97
B.1.1	Proofs of Propositions 3.1 and 3.3 . . . . .	97
B.1.2	Proofs of Proposition 3.2 and 3.4 . . . . .	98
B.1.3	Proof of Theorem 5 . . . . .	99
B.1.4	Proof of Theorem 6 . . . . .	101
B.1.5	Approximating expected prediction of classifiers . . . . .	103
B.2	Datasets . . . . .	103

<b>Appendix C Handling Missing Data in Decision Trees: A Probabilistic Approach . . .</b>	<b>106</b>
C.1 Proofs . . . . .	106
C.2 Expected Parameters Beyond Single Trees . . . . .	110
C.2.1 Forests . . . . .	110
C.2.2 Bagging and Random Forests . . . . .	111
C.2.3 Gradient Tree Boosting . . . . .	111
C.3 More Experiment Info . . . . .	113
<b>Appendix D Probabilistic Sufficient Explanations . . . . .</b>	<b>114</b>
D.1 Proof of Theorem 7 . . . . .	114
D.2 Beam Search Algorithm . . . . .	116
D.2.1 Pseudocode . . . . .	116
D.2.2 Computational Complexity . . . . .	116
D.3 More Experiment Details . . . . .	116
D.3.1 Computing Infrastructure . . . . .	116
D.3.2 Datasets and Preprocessing Steps . . . . .	117
D.3.3 Details on models . . . . .	117
D.3.4 Metric Calculation Details . . . . .	118
D.3.5 Effects of Parallelism on Performance . . . . .	119
D.3.6 More on SDP Lowerbounds . . . . .	119
D.3.7 Limitations . . . . .	120
D.3.8 More MNIST examples . . . . .	120
<b>Appendix E Why Is This an Outlier? Explaining Outliers by Submodular Optimization of Marginal Distributions . . . . .</b>	<b>122</b>

## LIST OF FIGURES

2.1	Logistic regression $\mathcal{F}(\mathbf{x}) = \text{sigmoid}(\mathbf{w}^T \mathbf{x})$ and two conformant naive Bayes models. . . . .	11
2.2	Standard image classification datasets: comparison of average cross entropy to the original predictions and classification accuracies between naive conformant learning (NaCL) and commonly used imputation methods. The conditional probabilities from NaCL are consistently closest to the full-data predictions, and NaCL consistently outperforms other methods with different percentages missing features. . . . .	15
2.3	Explanations for MNIST classifications. Grey features were not chosen as explanations; white/black are the true color of chosen features. From left to right: 1) all features; 2) support features; 3) top- $k$ support features; 4) sufficient explanation of size $k$ . . . . .	20
3.1	A vtree (a) over $\mathbf{X} = \{X_1, X_2, X_3\}$ and a generative and discriminative circuit pair (b, c) that conform to it. AND gates are colored as the vtree nodes they correspond to (blue and orange). For the discriminative circuit on the right, “hot wires” that form a path from input to output are colored red, for the given input configuration $\mathbf{x} = (X_1 = 1, X_2 = 0, X_3 = 0)$ . . . . .	26
3.2	Evaluating $\text{EC}_2$ for predictions under different percentages of missing features (x-axis) over four real-world <i>regression</i> datasets in terms of the RMSE (y-axis) of the predictions of $g((\mathbf{x}^m, \mathbf{x}^o))$ . Overall, exactly computing the expected predictions via $\text{EC}_2$ outperforms simple imputation schemes like median and mean as well as more sophisticated ones like MICE [Azur et al., 2011] or computing the MPE configuration with the PC $p$ . Detailed dataset statistics can be found in Appendix B.2. . . . .	35
3.3	Evaluating the first-order Taylor approximation $T_1(\sigma \circ g_m, p_n)$ of the expected predictions of a <i>classifier</i> for missing value imputation for different percentages of missing features (x-axis) in terms of the accuracy (y-axis). . . . .	38

4.1	Average test RMSE (y-axis, the lower, the better) on the Insurance data for different percentages of missing values (x-axis) when missingness is only at deployment time for a forest of 5 trees (left) or both at learning and deployment time for a single tree learned with XGBoost (right). For each experiment setting, we repeat 10 times and report the average error and their standard deviation. . . . .	48
5.1	Explanations for selected MNIST images. From left to right: 1) original image; 2) Anchors explanation; 3) our explanation with same number of features 4) our explanation with $k = 30$ . Gray pixels were not chosen for the explanation. Pixels chosen for the explanation are colored the same color as the original image. . . . .	65
5.2	For sufficient explanations (sizes 1-30), we plot SDP estimates (green) vs SDP lower bounds calculated based on expected log-odds (blue), and lower bounds based on approximate expected prediction (red), averaged over 50 test images of MNIST. Shaded regions represent one standard deviation. . . . .	67
5.3	Tradeoff between expected prediction and marginal probability for MLSEs. The first plot is for positive label images (5s); the second is for negative label images (3s). Expected predictions were computed using a first order approximation. . . . .	69
6.1	Effective outlier feature detection with inlier maximization with a fixed budget. Each row corresponds to (top to bottom): original images, corrupted images, outliers found with budget $k = 60$ (denoted with red pixels), and images that have been cleaned by the generative model. Each column corresponds to a different corruption process (left to right): Square(3x3), Square(5x5), Square(7x7), Square(9x9), Smiley, Column(14), Row(14), Random(120), Random(200). . . . .	84
B.1	A vtree (a) over $\mathbf{X} = \{X_1, X_2, X_3\}$ and corresponding circuits that are respectively equivalent to a given Logistic Regression model with parameters $w_0, w_1, w_2, w_3$ , and a Naive Bayes model with parameters $\theta_c, \theta_{x_i c}, \theta_{x_i \neg c}$ . . . . .	101

D.1 MNIST Runtimes. Average cumulative time taken until completion of each iteration of the beam search algorithm for different numbers of threads. . . . . 118

D.2 More MNIST examples along with explanations with cardinality constraint  $k = 10, 20, 30$ . 121

## LIST OF TABLES

2.1	Summary of our testbed. . . . .	15
2.2	Three unbalanced UCI datasets with categorical features: comparison of average cross entropy to the original predictions between naive conformant learning (NaCL) and commonly used imputation methods. The closest are denoted in bold. . . . .	16
2.3	Three unbalanced UCI datasets with categorical features: comparison of weighted F1 scores between naive conformant learning (NaCL) and commonly used imputation. The highest are denoted in bold. . . . .	16
5.1	Comparison of average expected log-odds, SDP, and marginals between Anchors and MLSE, averaged over 50 random MNIST test images. We take the absolute value of $EP_{\mathcal{O}}(z)$ to measure confidence of the explanations (since it could be negative). $MLSE_s$ sets the cardinality constraint to the same size of the Anchors explanation for each image. The $\pm$ denotes one standard deviation. The SDP values are approximated. . . .	66
5.2	SDP, marginal probability, and size statistics for explanations found using different methods for the adult dataset. . . . .	68
6.1	Datasets . . . . .	85
6.2	Percentage and Avg Outliers found for Maximizing Marginals of Inliers ( $k = 60$ ) . . .	86
6.3	Percentage and Avg Outliers found for Minimizing Marginals of Outliers ( $k = 60$ ) . .	87
6.4	Accuracy Gains for Maximizing Marginals of Inliers ( $k = 60$ ) . . . . .	88
6.5	Accuracy Gains for Minimizing Marginals of Outliers ( $k = 60$ ) . . . . .	88
6.6	Outlier Aspect Mining Baselines. Each dataset has 3 labeled ground truth. Each row compares the F1 score across different methods for the corresponding outlier ground truth. . . . .	89

6.7	Summary of Table 6.6 Results. Each entry in the table denotes how often does the method get better F1 results than others. . . . .	90
B.1	Statistics as number of train, validation and test samples and features (after discretization) for the datasets employed in the regression (top half) and classification (bottom half) experiments. . . . .	105
B.2	Statistics on the runtime of our algorithm versus MICE and the Monte Carlo Sampling algorithm. The reported times for prediction times are for one configuration of the experiment. As we tried 10 different missingness percentages and repeated each 10 times, the total time of experiment is 100 times the value in the table. Learning time refers to learning the generative circuit and is done only once. . . . .	105
C.1	Statistics about the datasets used in the experiments. . . . .	113

## ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor, Professor Guy Van den Broeck. You have not only been a great advisor and mentor but have also created a nurturing lab environment that has significantly enriched my academic journey. I am thankful for the freedom you've given me to focus on topics of personal interest, all while encouraging me to tackle more challenging and impactful projects. Over the past few years, I have undoubtedly learned a great deal from you and will genuinely miss our spontaneous whiteboard discussions with our fellow labmates.

I would also like to express my gratitude to Professors Quanquan Gu, Bolei Zhou, and Aditya Grover for being members of my dissertation committee. I appreciate you attending my presentations and providing valuable feedback during our discussions.

The members of StarAI++ lab and UCLA friends has been both sources of fun and inspiration: YooJung, Antonio, Tal, Steven, Yujia, Arthur, Andy, Anton, Kareem, Yitao, Honghua, Meihua, Zhe, Anji, Poorva, Nikil, William, Daniel, Ellie, Eric, Paolo, Laura, Aishwarya, Chi, Jieyu, Atefeh, Farnaz, Ana, and Omead (ordering sampled from an undisclosed distribution). We shared different aspects of our academic journey together, but one thing in common is that each of you augmented my PhD journey in a positive manner.

I also would like to thank my industry mentors: Mori, Arjun, Weinan, Xiolin, Diana, Humberto, Sishi, Albert, Naz, Prithu (chronological order). Thanks for giving me opportunities to try different things while honing both my engineering and research skills.

I extend my heartfelt gratitude to my brother Pooya and dear friends: Navid, Shain, Sepehr, Kevin, Linh, Karthik, Sasha, Omead, and Tara (random order), who have been unwavering sources of support throughout my journey in graduate school. From our spontaneous group chats to memorable video game sessions and delightful hangouts, you have played an indispensable role in keeping me grounded and has made the demanding path of graduate school more manageable.

Finally, special thanks to my parents for their infinite love and support, without it none of this would have been possible.



## VITA

2015–2018 B.S. in Computer Science, *summa cum laude*. University of California, Irvine.

## PUBLICATIONS

**Pasha Khosravi**, Yitao Liang, YooJung Choi, Guy Van den Broeck. What to Expect of Classifiers? Reasoning about Logistic Regression with Missing Features. *In Proceedings of International Joint Conferences on Artificial Intelligence (IJCAI), 2019.*

**Pasha Khosravi**, YooJung Choi, Yitao Liang, Antonio Vergari and Guy Van den Broeck. On Tractable Computation of Expected Predictions. *In Proceedings of NeurIPS, 2019.*

**Pasha Khosravi**, Antonio Vergari, YooJung Choi, Yitao Liang and Guy Van den Broeck. Handling Missing Data in Decision Trees: A Probabilistic Approach. *In The Art of Learning with Missing Values Workshop at ICML (Artemiss), 2020.*

Meihua Dang, **Pasha Khosravi**, Yitao Liang, Antonio Vergari, Guy Van den Broeck. Juice: A Julia Package for Logic and Probabilistic Circuits. *In Proceedings of AAAI Conference (Demo Track), 2021.*

Eric Wang, **Pasha Khosravi**, Guy Van den Broeck. Probabilistic Sufficient Explanations. *In Proceedings of International Joint Conferences on Artificial Intelligence (IJCAI), 2021.*

**Pasha Khosravi**, Antonio Vergari, Guy Van den Broeck. Why Is This an Outlier? Explaining Outliers by Submodular Optimization of Marginal Distributions. *In the 5th Workshop on Tractable Probabilistic Modeling at UAI (TPM 2022)*.

# CHAPTER 1

## Introduction

The prevalence of Machine Learning models in our daily lives continues to rise. Yet, persistent challenges like missing data, outliers, and the complex behaviors of these models pose obstacles to their reliability and utility. In this work, we explore the role probabilistic reasoning can play in enhancing trustworthiness of machine learning models. From effectively handling missing data and providing reliable explanations for classifications to understanding and addressing outliers, we delve into the potential of tractable probabilistic approaches in addressing these challenges.

The first half of the thesis is dedicated to addressing the issue of missing values in various family of models. In practice, encountering missing data is very common and could happen for many different reasons. A big challenge with missing data is that most machine learning models are inherently not equipped to handle missing values. A common approach to mitigate missing data involves imputation, which is the process of replacing the missing values with some reasonable estimates. However, even with accurate imputations, we are we are still neglecting some of the uncertainty that comes from alternative ways of filling out the missing values. That's where concept of Expected Prediction (EXP) comes into play. In an ideal world, we would consider all potential ways of filling out missing data, evaluate our model's outcome across those cases, and then take a weighted average of the outcomes based on how likely each scenario was.

We show that this ideal goal can be achieved to some extent. Firstly, we need a good estimate of how likely each scenario is, that's where tractable probabilistic model's such as probabilistic circuits come in. They allow learning expressive density estimations from data, and additionally enable many tractable queries to leverage the density estimation model.

Of course, we cannot afford to actually evaluate our model on all numerous configurations of filling missing values. Even in simple case of binary features, the number of possible ways to fill out missing values grows exponentially. Therefore, we started by systematically studying computation complexity of Expected Prediction (EXP) framework for many family of models. We proved some hardness results (and provided workaround for those cases), and came up with tractable algorithms computing EXP for some family of models. Chapter 2 introduces NaCL for a way of handling missing data in logistic regression using their close relation to Naive Bayes models. Chapter 3 explores computation challenges of EXP for pair of generative and discriminative circuits. Finally, Chapter 4 provides method for computing EXP for decision trees. In all these cases, we showcased that computing EXP can lead to better performance while also being competitive in runtime.

In the world of data driven decision making, one of the most ubiquitous uses of machine learning in our daily lives is through classification. They enrich our lives by automating decisions like detecting spam email, evaluating credit-worthiness for loan applicants to diagnosing patients. However, these model do not behave as expected, and the more sensitive the use-case the less we want to rely on complex and non-interpretable outcomes of classifiers. Hence, there is a need to explain the decision making of classifiers.

Hence, there has been many different approaches to explaining outcomes of classifiers. On one side, many explanation methods make strong assumptions such as independence of features or ignore correlations between features. On another side, logical explanation methods demand guarantees that lead to complex explanations. In Chapter 5 we introduced Probabilistic Sufficient Explanations to try to balance between these two approaches. We formulate the problem as choosing a small subset of features that if only observing those features we can Sufficiently conclude, with probabilistic guarantees, that get the same outcome as observing all the features. We used expected prediction as guiding factor of finding the best subset that provides these guarantees.

Outliers are another phenomenon that hinder machine learning models. There has been many studies on how to detect outliers. We wanted to go an step further and explain why an instance was an outlier. In Chapter 6, we leverage supermodularity of marginal probability distributions to

accelerate the search for find the subset of features contributed to an instance being an outlier in high dimensions.

## CHAPTER 2

### Reasoning about Logistic Regression with Missing Features

While discriminative classifiers often yield strong predictive performance, missing feature values at prediction time can still be a challenge. Classifiers may not behave as expected under certain ways of substituting the missing values, since they inherently make assumptions about the data distribution they were trained on. In this chapter, we propose a novel framework that classifies examples with missing features by computing the expected prediction with respect to a feature distribution. Moreover, we use geometric programming to learn a naive Bayes distribution that embeds a given logistic regression classifier and can efficiently take its expected predictions. Empirical evaluations show that our model achieves the same performance as the logistic regression with all features observed, and outperforms standard imputation techniques when features go missing during prediction time. Furthermore, we demonstrate that our method can be used to generate “sufficient explanations” of logistic regression classifications, by removing features that do not affect the classification.

#### 2.1 Introduction

Missing values are pervasive in real-world machine learning applications. Learned classifiers usually assume all input features are known, but when a classifier is deployed, some features may not be available, or too difficult to acquire. This can be due to the noisy nature of our environment, unreliable or costly sensors, and numerous other challenges in gathering and managing data [Dekel and Shamir, 2008; Graham, 2012]. Consider autonomous driving for example, where blocked sensors may leave observations incomplete.

Moreover, it can be difficult to anticipate the many ways in which a learned classifier will be deployed. For example, the same classifier could be used by different doctors who choose to run different medical tests on their patients, and this information is not known at learning time. Nevertheless, even a small portion of missing data can severely affect the performance of well-trained models, leading to predictions that are very different from the ones made when all data is observed.

Certain machine learning models, such as probabilistic graphical models, provide a natural solution to missing features at prediction time, by formulating the problem as a probabilistic inference task [Koller and Friedman, 2009; Darwiche, 2009b]. Unfortunately, with the increasing emphasis on predictive performance, the general consensus is that such generative models are not competitive as classifiers given fully-observed feature vectors, and that discriminatively-trained models are to be preferred [Ng and Jordan, 2002].

To alleviate the impact of missing data on discriminative classifiers, it is common practice to substitute the missing values with plausible ones [Schafer, 1999; Little and Rubin, 2014]. As we will argue later, a drawback for such imputation techniques is that they can make overly strong assumptions about the feature distribution. Furthermore, to be compatible with many types of classifiers, they tend to overlook how the multitude of possible imputed values would interact with the classifier at hand, and risk yielding biased predictions.

To better address this issue, we propose a principled framework of handling missing features by reasoning about the classifier’s *expected* output given the feature distribution. One obvious advantage is that it can be tailored to the given family of classifiers and feature distributions. In contrast, the popular mean imputation approach only coincides with the expected prediction for simple models (e.g. linear functions) under very strong independence assumptions about the feature distribution. We later show that calculating the expected predictions with respect to arbitrary feature distributions is computationally highly intractable. In order to make our framework more feasible in practice, we leverage generative-discriminative counterpart relationships to learn a joint distribution that can take expectations of its corresponding discriminative classifier. We call this problem

*conformant learning*. Then, we develop an algorithm, based on geometric programming, for a well-known example of such relationship: naive Bayes and logistic regression. We call this specific algorithm naive conformant learning (NaCL).

Through an extensive empirical evaluation over five characteristically distinct datasets, we show that NaCL consistently achieves better estimates of the conditional probability, as measured by average cross entropy and classification accuracy, compared to commonly used imputation methods. Lastly, we conduct a short case study on how our framework can be applied to *explain* classifications.

## 2.2 The Expected Prediction Task

In this section, we describe our intuitive approach to making a prediction when features are missing and discuss how it relates to existing imputation methods. Then we study the computational hardness of our expected prediction task.

We use uppercase letters to denote features/variables and lowercase letters for their assignment. Sets of variables  $\mathbf{X}$  and their joint assignments  $\mathbf{x}$  are written in bold. For an assignment  $x$  to a binary variable  $X$ , we let  $\bar{x}$  denote its negation. Concatenation  $\mathbf{XY}$  denotes the union of disjoint sets. The set of all possible assignments to  $\mathbf{X}$  is denoted  $\mathcal{X}$ .

Suppose we have a model trained with features  $\mathbf{X}$  but are now faced with the challenge of making a prediction without knowing all values  $\mathbf{x}$ . In this situation, a common solution is to impute certain substitute values for the missing data (for example their mean) [Little and Rubin, 2014]. However, the features that were observed provide information not only about the class but also about the missing features, yet this information is typically not taken into account by popular methods such as mean imputation.

We propose a very natural alternative: to utilize the feature distribution to probabilistically reason about what a predictor is expected to return if it could observe the missing features.



**Definition 1.** Let  $\mathcal{F} : \mathcal{X} \rightarrow \mathbb{R}$  be a predictor and  $P$  be a distribution over features  $\mathbf{X}$ . Given a partitioning of features  $\mathbf{X} = \mathbf{Y}\mathbf{M}$  and an assignment  $\mathbf{y}$  to some of the features  $\mathbf{Y}$ , the expected prediction task is to compute

$$E_{\mathcal{F},P}(\mathbf{y}) = \mathbb{E}_{\mathbf{m} \sim P(\mathbf{M}|\mathbf{y})} [\mathcal{F}(\mathbf{y}\mathbf{m})].$$

The expected prediction task and (mean) imputation are related, but only under very restrictive assumptions.

**Example 1.** Let  $\mathcal{F} : \mathcal{X} \rightarrow \mathbb{R}$  be a linear function. That is,  $\mathcal{F}(\mathbf{x}) = \sum_{x \in \mathbf{X}} w_X x$  for some weights  $\mathbf{w}$ . Suppose  $P$  is a distribution over  $\mathbf{X}$  that assumes independence between features:  $P(\mathbf{X}) = \prod_{X \in \mathbf{X}} P(X)$ . Then, using linearity of expectation, the following holds for any partial observation  $\mathbf{y}$ :

$$\begin{aligned} E_{\mathcal{F},P}(\mathbf{y}) &= \mathbb{E}_{\mathbf{m} \sim P(\mathbf{M}|\mathbf{y})} \left[ \sum_{y \in \mathbf{Y}} w_Y y + \sum_{m \in \mathbf{M}} w_M m \right] \\ &= \sum_{y \in \mathbf{Y}} w_Y y + \sum_{M \in \mathbf{M}} w_M \mathbb{E}_{m \sim P(M)} [m]. \end{aligned}$$

Hence, substituting the missing features with their means effectively computes the expected predictions of linear models if the independence assumption holds. Furthermore, if  $\mathcal{F}$  is the true conditional probability of the labels and features are generated by a fully-factorized  $P(\mathbf{X})$ , then classifying by comparing the expected prediction  $E_{\mathcal{F},P}$  to 0.5 is Bayes optimal on the observed features. That is, an expected prediction higher than 0.5 means that the positive class is the most likely one given the observation, and thus minimizes expected loss, according to the distribution defined by  $\mathcal{F}$  and  $P$ .

**Example 2.** Consider a logistic regression model  $\mathcal{G}(\mathbf{x}) = \text{sigmoid}(\mathcal{F}(\mathbf{x}))$  where  $\mathcal{F}$  is a linear function. Now, mean imputation no longer computes the expected prediction, even when the independence assumption in the previous example holds. In particular, if  $\mathbf{y}$  is a partial observation such that  $\mathcal{G}(\mathbf{y}\mathbf{m})$  is positive for all  $\mathbf{m}$ , then the mean-imputed prediction is an over-approximation

of the expected prediction:

$$\mathcal{G}(\mathbf{y} \mathbb{E}[\mathbf{m}]) = \text{sigmoid}(\mathbb{E}[\mathcal{F}(\mathbf{y}\mathbf{m})]) > \mathbb{E}[\text{sigmoid}(\mathcal{F}(\mathbf{y}\mathbf{m}))] = E_{\mathcal{G},P}(\mathbf{y}).$$

This is due to Jensen’s inequality and concavity of the sigmoid function in the positive portion; conversely, it is an under-approximation in the negative cases.

Example 1 showed how to efficiently take the expectation of a linear function w.r.t. a fully factorized distribution. Unfortunately, the expected prediction task is in general computationally hard, even on simple classifiers and distributions.

**Proposition 2.1.** *Taking expectation of a nontrivial classifier w.r.t. a uniform distribution is #P-hard.*

*Proof.* Suppose our classifier tests whether a logical constraint holds between the input features. Then asking whether there exists a positive example is equivalent to SAT which is NP-hard. The expected classification on a uniform distribution is solving an even harder task, of counting solutions to the constraint, which is #P-hard [Roth, 1996b].  $\square$

Next, consider the converse in which the classifier is trivial but the distribution is more general.

**Proposition 2.2.** *The expectation of a classifier that returns the value of a single feature w.r.t. a distribution represented by a probabilistic graphical model is #P-hard.*

*Proof.* Computing expectations of such classifier is as hard as computing marginals in the feature distribution, which is #P-hard for graphical models [Roth, 1996b].  $\square$

Previous propositions showed that the expected prediction task stays intractable, even when we allow either the distribution or the classifier to be trivial.

Our next theorem states that the task is hard even for a relatively simple classifier and a tractable distribution.<sup>1</sup>

---

<sup>1</sup>All proofs can be found in Appendix A.1.

**Theorem 1.** *Computing the expectation of a logistic regression classifier over a naive Bayes distribution is NP-hard.*

That is, the expected prediction task is hard even though logistic regression classification and probabilistic reasoning on naive Bayes models can both be done in linear time.

In summary, while the expected prediction task appears natural for dealing with missing data, its vast intractability poses a serious challenge, especially compared to efficient alternatives such as imputation. Next, we investigate specific ways of practically overcoming this challenge.

### 2.3 Joint Distributions as Classifiers

As shown previously, taking expectations is intractable for arbitrary pairs of classifiers and distributions. In this section, we propose *conformant learning* which aims to learn a joint distribution that encodes a given classifier as well as a feature distribution. On such distribution, the expected prediction task is well-defined as probabilistic inference, and is tractable for a large class of probabilistic models, including naive Bayes [Darwiche, 2009b; Dechter, 2013].

We first describe how a joint distribution can be used as a classifier that inherently support missing features during prediction time. Given a distribution  $P(\mathbf{X}, C)$  over the features  $\mathbf{X}$  and class variable  $C$ , we can classify a partial observation  $\mathbf{y}$  simply by computing the conditional probability  $P(c | \mathbf{y})$  where  $c$  denotes the positive class.<sup>2</sup> In some sense, a joint distribution embeds a classifier  $P(C | \mathbf{Y})$  for each subset of observed features  $\mathbf{Y}$ . In fact, computing  $P(c | \mathbf{y})$  is *equivalent to computing the expected prediction* of classifier  $\mathcal{F}$  that outputs  $P(c | \mathbf{x})$  for every  $\mathbf{x}$ , with respect to distribution  $P(\mathbf{X})$ :

$$P(c | \mathbf{y}) = \sum_{\mathbf{m}} P(c, \mathbf{m} | \mathbf{y}) = \sum_{\mathbf{m}} P(c | \mathbf{m}\mathbf{y})P(\mathbf{m} | \mathbf{y})$$

---

<sup>2</sup>We assume binary class for conciseness, but our approach easily generalizes to multiclass. Details can be found in Appendix A.2.

$$= \mathbb{E}_{\mathbf{m} \sim P(\mathbf{M}|\mathbf{y})} [P(c|\mathbf{y}\mathbf{m})] = E_{\mathcal{F},P}(\mathbf{y}). \quad (2.3.1)$$

Nevertheless, the prevailing consensus is that in practice discriminatively training a classifier  $P(C|\mathbf{X})$  should be preferred to generatively learning  $P(\mathbf{X}, C)$ , because it tends to achieve higher classification accuracy [Bouchard and Triggs, 2004; Ulusoy and Bishop, 2005].

There are many generative-discriminative pairs obtained from fitting the same family of probabilistic models to optimize either the joint or conditional likelihood [Jaakkola and Haussler, 1999; Sutton and McCallum, 2012; Liang and Van den Broeck, 2019b], including naive Bayes and logistic regression [Ng and Jordan, 2002]. We formally describe such relationship as follows:

**Definition 2.** We say  $P(\mathbf{X}, C)$  conforms with  $\mathcal{F} : \mathcal{X} \rightarrow [0, 1]$  if their classifications always agree, i.e.:

$$\forall \mathbf{x} \quad P(c|\mathbf{x}) = \mathcal{F}(\mathbf{x})$$

Next, let us study naive Bayes models in detail as they support efficient inference and thus are a good target distribution to leverage for the expected prediction task. Naive Bayes models assume that features are mutually independent given the class; that is, its joint distribution is  $P(\mathbf{X}, C) = P(C) \prod_{X \in \mathbf{X}} P(X|C)$ . Under such assumption, marginal inference takes linear time, and so does computing expectations under missing features as in Equation 2.3.1.

Logistic regression is the discriminative counterpart to naive Bayes. It has parameters  $\mathbf{w}$  and posits that <sup>3</sup>

$$\mathcal{F}(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T \cdot \mathbf{x}}}.$$

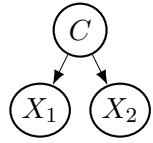
Any naive Bayes classifier can be translated to an equivalent logistic regression classifier on fully observed features.

---

<sup>3</sup>Here,  $\mathbf{x}$  also includes a dummy feature that is always 1 to correspond with the bias parameter  $w_0$ .

$$\mathbf{w} = \begin{bmatrix} -1.16 \\ 2.23 \\ -0.20 \end{bmatrix} \quad \begin{array}{|c|c|c|} \hline X_1 & X_2 & \mathcal{F}(x_1, x_2) \\ \hline 1 & 1 & 0.70 \\ 1 & 0 & 0.74 \\ 0 & 1 & 0.20 \\ 0 & 0 & 0.24 \\ \hline \end{array}$$

(a) Logistic regression weights and resulting predictions.



$P_1(c)$	$C$	$P_1(x_1 C)$	$C$	$P_1(x_2 C)$
0.5	1	0.8	1	0.45
	0	0.3	0	0.5
$P_2(c)$	$C$	$P_2(x_1 C)$	$C$	$P_2(x_2 C)$
0.36	1	0.6	1	0.9
	0	0.14	0	0.92

(b) Two naive Bayes distributions with same structure.

Figure 2.1: Logistic regression  $\mathcal{F}(\mathbf{x}) = \text{sigmoid}(\mathbf{w}^T \mathbf{x})$  and two conformant naive Bayes models.

**Lemma 2.** *Given a naive Bayes distribution  $P$ , there is a unique logistic regression model  $\mathcal{F}$  that it conforms with. Such logistic regression model has the following weights:*

$$w_0 = \log \frac{P(c)}{P(\bar{c})} + \sum_{i=1}^n \log \frac{P(\bar{x}_i | c)}{P(\bar{x}_i | \bar{c})}$$

$$w_i = \log \frac{P(x_i | c)}{P(x_i | \bar{c})} \cdot \frac{P(\bar{x}_i | \bar{c})}{P(\bar{x}_i | c)}, \quad i = 1, \dots, n$$

Here,  $x, \bar{x}$  denote  $X=1, X=0$  respectively.

The lemma above can be drawn from Roos et al. [2005] and Ng and Jordan [2002]. Complete proofs of all lemmas can be found in Appendix A.1.

Consider for example the naive Bayes (NB) distribution  $P_1$  in Figure 2.1b. For all possible feature observations, the NB classification  $P_1(c | \mathbf{x})$  is equal to that of logistic regression (LR)  $\mathcal{F}$  in Figure 2.1a, whose weights are as given by above lemma (i.e.,  $P_1$  conforms with  $\mathcal{F}$ ). Furthermore, distribution  $P_2$  also translates into the same logistic regression. In fact, there can be infinitely many such naive Bayes distributions.

**Lemma 3.** *Given a logistic regression  $\mathcal{F}$  and  $\theta \in (0, 1)^n$ , there exists a unique naive Bayes model  $P$  such that*

$$\begin{aligned} P(c | \mathbf{x}) &= \mathcal{F}(\mathbf{x}), \quad \forall \mathbf{x} \\ P(x_i | c) &= \theta_i, \quad i = 1, \dots, n. \end{aligned}$$

That is, given a logistic regression there are infinitely many naive Bayes models that conform with it. Moreover, after fixing values for  $n$  parameters of the NB model, there is a uniquely corresponding naive Bayes model.

We can expect this phenomenon to generalize to other generative-discriminative pairs; given a conditional distribution  $P(C | \mathbf{X})$  there are many possible feature distributions  $P(\mathbf{X})$  to define a joint distribution  $P(\mathbf{X}, C)$ . For instance, distributions  $P_1$  and  $P_2$  in Figure 2.1b assign different probabilities to feature observations;  $P_1(\bar{x}_1, \bar{x}_2) = 0.23$  whereas  $P_2(\bar{x}_1, \bar{x}_2) = 0.06$ . Hence, we wish to define which one of these models is the “best”. Naturally, we choose the one that best explains a given dataset of feature observations.<sup>4</sup>

**Definition 3.** *Let  $\mathcal{F} : \mathcal{X} \rightarrow [0, 1]$  be a discriminative classifier and  $D$  be a dataset where each example  $d$  is a joint assignment to  $\mathbf{X}$ . Given a family of distributions over  $C$  and  $\mathbf{X}$ , let  $\mathcal{P}$  denote the subset of them that conforms with  $\mathcal{F}$ . Then conformant learning on  $D$  is to solve the following optimization:*

$$\arg \max_{P \in \mathcal{P}} \prod_{d \in D} P(d) = \arg \max_{P \in \mathcal{P}} \prod_{d=(\mathbf{x}) \in D} \sum_c P(\mathbf{x}, c). \quad (2.3.2)$$

The learned model thus conforms with  $\mathcal{F}$  and defines a feature distribution; therefore, we can take the expectation of  $\mathcal{F}$  via probabilistic inference. In other words, it attains the desired classification performance of the given discriminative model while also returning sophisticated predictions under missing features. Specifically, conformant naive Bayes models can be used to

---

<sup>4</sup>Here we assume i.i.d. sampled data. If a true distribution is known, we can equivalently minimize the KL-divergence to it.

efficiently take expectations of logistic regression classifiers. Note that this does not contradict Theorem 1 which considers arbitrary pairs of LR and NB models.

## 2.4 Naive Conformant Learning

In this section, we study a special case of conformant learning – naive conformant learning (NaCL), and show how it can be solved as a geometric program.

A naive Bayes distribution is defined by a parameter set  $\theta$  that consists of  $\theta_c, \theta_{\bar{c}}$ , and  $\theta_{x|c}, \theta_{x|\bar{c}}$  for all  $x$ . *Naive conformant learning* outputs the naive Bayes distribution  $P_\theta$  that maximizes the (marginal) likelihood given the dataset and conforms with a given logistic regression model  $\mathcal{F}$ .

We will next show that above problem can be formulated as a *geometric program*, an optimization problem of the form:

$$\begin{aligned} \min & f_0(x) \\ \text{s.t.} & f_i(x) \leq 1, \quad i = 1 \dots m \\ & g_i(x) = 1, \quad i = 1 \dots p \end{aligned}$$

where each  $f_i$  is a posynomial and  $g_i$  monomial. A *monomial* is a function of the form  $bx_1^{a_1} \dots x_n^{a_n}$  defined over positive real variables  $x_1, \dots, x_n$  where  $b > 0$  and  $a_i \in \mathbb{R}$ . A *posynomial* is a sum of monomials. Every geometric program can be transformed into an equivalent convex program through change of variables, and thus its global optimum can be found efficiently [Boyd et al., 2007].

To maximize the likelihood, we instead minimize its inverse. Let  $n(\mathbf{x})$  denote the number of times an assignment  $\mathbf{x}$  appears in dataset  $D$ . Then the objective function is:

$$\prod_{d \in D} P_\theta(d)^{-1} = \prod_{\mathbf{x}} P_\theta(\mathbf{x})^{-n(\mathbf{x})} = \prod_{\mathbf{x}} \left( \sum_c \prod_{x \in \mathbf{x}} \theta_{x|c} \theta_c \right)^{-n(\mathbf{x})}.$$

Above formula, directly expanded, is not a posynomial. In order to express it as a posynomial we consider an auxiliary dataset  $D'$  constructed from  $D$  as follows: for each data point  $d_j = (\mathbf{x}) \in D$ , there are  $d'_{j,c} = (\mathbf{x}, c) \in D'$  with weight  $\alpha_{j,c}$  for all values of  $c$ . If the weights are such that  $\alpha_{j,c} \geq 0$  and  $\sum_c \alpha_{j,c} = 1$  for all  $j$ , then the inverse of the expected joint likelihood given the new dataset  $D'$  is

$$\begin{aligned} \prod_{d'_{j,c} = (\mathbf{x}, c) \in D'} P_\theta(\mathbf{x}, c)^{-\alpha_{j,c}} &= \prod_{d_j = (\mathbf{x}) \in D} \prod_c P_\theta(\mathbf{x})^{-\alpha_{j,c}} P_\theta(c | \mathbf{x})^{-\alpha_{j,c}} \\ &= \prod_{d \in D} P_\theta(d)^{-1} \cdot \prod_{d_j = (\mathbf{x}) \in D, c} P_\theta(c | \mathbf{x})^{-\alpha_{j,c}}. \end{aligned} \quad (2.4.1)$$

For any  $P_\theta \in \mathcal{P}$ , the conditional distribution  $P_\theta(C | \mathbf{X})$  is fixed by the logistic regression model; in other words, the last product term in Equation 2.4.1 is a constant. Therefore, maximizing the expected (joint) likelihood on a completed dataset must also maximize the marginal likelihood, which is our original objective. Intuitively, maximizing the joint likelihood on any dataset is equivalent to maximizing the marginal likelihood  $P(\mathbf{X})$  if the conditional distribution  $P(C | \mathbf{X})$  is fixed. Now our objective function can be written as a monomial in terms of the parameters:

$$\prod_{d'_{j,c} \in D'} P_\theta(d'_{j,c})^{-\alpha_{j,c}} = \prod_{d'_{j,c} = (\mathbf{x}, c) \in D'} \left( \theta_c \prod_{x \in \mathbf{x}} \theta_{x|c} \right)^{-\alpha_{j,c}}. \quad (2.4.2)$$

Next, we express the set of conformant naive Bayes distributions  $\mathcal{P}$  as geometric program constraints in terms of  $\theta$ . An NB model  $P_\theta$  conforms with an LR  $\mathcal{F}$  if and only if its corresponding logistic regression weights, according to Lemma 2, match those of  $\mathcal{F}$ . Hence,  $P_\theta \in \mathcal{P}$  precisely when

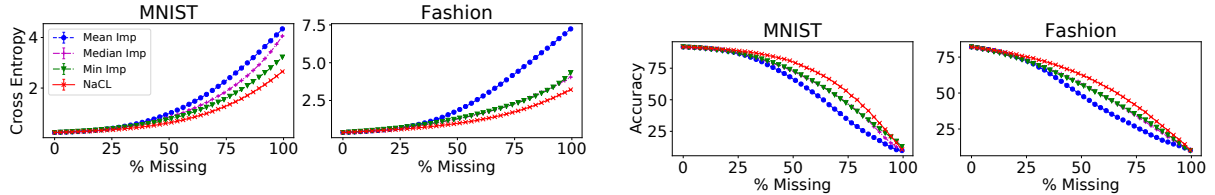
$$e^{w_0} \theta_c^{-1} \theta_{\bar{c}} \prod_{i=1}^n \theta_{\bar{x}_i|c}^{-1} \theta_{\bar{x}_i|\bar{c}} = 1 \quad (2.4.3)$$

$$e^{w_i} \theta_{x_i|c}^{-1} \theta_{x_i|\bar{c}} \theta_{\bar{x}_i|c} \theta_{\bar{x}_i|\bar{c}}^{-1} = 1, \quad \forall i \quad (2.4.4)$$



DATASETS	SIZE	# CLASSES: DIST.	# FEATURES	FEATURE TYPES
MNIST	60K	10: BALANCED	784	INT. PIXEL VALUE
FASHION	60K	10: BALANCED	784	INT. PIXEL VALUE
COVTYPE	581K	7: UNBALANCED	54	CONT. & CATEGORICAL
ADULT	49K	2: UNBALANCED	14	INT. & CATEGORICAL
SPLICE	3K	3: UNBALANCED	61	CATEGORICAL

Table 2.1: Summary of our testbed.



(a) Cross Entropy (fidelity to logistic regression predictions)

(b) Accuracy (impact on test-set prediction quality)

Figure 2.2: Standard image classification datasets: comparison of average cross entropy to the original predictions and classification accuracies between naive conformant learning (NaCL) and commonly used imputation methods. The conditional probabilities from NaCL are consistently closest to the full-data predictions, and NaCL consistently outperforms other methods with different percentages missing features.

We also need to ensure that the parameters define a valid probability distribution (e.g.,  $\theta_c + \theta_{\bar{c}} = 1$ ). Because such posynomial equalities are not valid geometric program constraints, we instead relax them to posynomial inequalities:<sup>5</sup>

$$\theta_c + \theta_{\bar{c}} \leq 1, \theta_{x_i|c} + \theta_{\bar{x}_i|c} \leq 1, \theta_{x_i|\bar{c}} + \theta_{\bar{x}_i|\bar{c}} \leq 1, \forall i \quad (2.4.5)$$

Putting everything together, naive conformant learning can be solved as a geometric program whose objective function is given by Equation 2.4.2 and constraints by Equations 2.4.3 – 2.4.5. We used the GPkit<sup>6</sup> library to solve our geometric programs.

<sup>5</sup>The learned parameters may not sum to 1. They can still be interpreted as a multi-valued NB with same likelihood that conforms with  $\mathcal{F}$ . These constraints were always active in our experiments.

<sup>6</sup><https://gpkit.readthedocs.io>

CROSS ENTROPY	COVTYPE				ADULT				SPLICE			
	20%	40%	60%	80%	20%	40%	60%	80%	20%	40%	60%	80%
UNDER % MISSING												
MIN IMPUTATION	12.8	15.5	20.7	29.4	41.0	49.2	55.4	59.6	71.3	81.8	97.2	117.2
MAX IMPUTATION	52.6	89.1	133.5	187.7	84.2	114.6	125.3	114.5	70.5	78.3	89.3	103.2
MEAN IMPUTATION	12.8	15.6	21.2	30.5	34.1	38.7	44.8	52.6	69.2	<b>74.7</b>	<b>82.4</b>	92.3
MEDIAN IMPUTATION	12.8	15.7	21.4	30.8	35.3	41.2	48.6	57.8	70.0	75.7	83.0	<b>92.0</b>
NAIVE CONFORMANT LEARNING	<b>12.6</b>	<b>14.8</b>	<b>18.9</b>	<b>25.8</b>	<b>33.6</b>	<b>37.0</b>	<b>41.2</b>	<b>46.6</b>	<b>69.1</b>	<b>74.7</b>	82.8	94.0

Table 2.2: Three unbalanced UCI datasets with categorical features: comparison of average cross entropy to the original predictions between naive conformant learning (NaCL) and commonly used imputation methods. The closest are denoted in bold.

WEIGHTED F1	COVTYPE				ADULT				SPLICE			
	20%	40%	60%	80%	20%	40%	60%	80%	20%	40%	60%	80%
UNDER % MISSING												
MIN IMPUTATION	64.0	58.1	52.2	46.1	81.7	79.3	77.5	<b>76.0</b>	86.9	69.8	49.2	38.8
MAX IMPUTATION	49.8	44.4	41.6	37.3	81.7	79.3	77.4	<b>76.0</b>	86.9	69.8	49.1	38.8
MEAN IMPUTATION	64.0	58.0	52.2	46.3	82.9	79.8	75.3	70.7	91.8	82.3	66.2	45.7
MEDIAN IMPUTATION	64.0	58.1	52.2	46.1	82.7	79.2	74.8	70.5	89.4	77.6	59.5	42.5
NAIVE CONFORMANT LEARNING	<b>66.1</b>	<b>61.7</b>	<b>56.9</b>	<b>51.7</b>	<b>83.4</b>	<b>81.2</b>	<b>77.9</b>	73.5	<b>93.3</b>	<b>87.2</b>	<b>76.6</b>	<b>59.1</b>

Table 2.3: Three unbalanced UCI datasets with categorical features: comparison of weighted F1 scores between naive conformant learning (NaCL) and commonly used imputation. The highest are denoted in bold.

## 2.5 Empirical Evaluation

In this section, we empirically evaluate the performance of naive conformant learning (NaCL) and provide a detailed discussion of our method’s advantages over existing imputation approaches in practice. More specifically, we want to answer the following questions:

**Q1** Does NaCL reliably estimate the probabilities of the original logistic regression with full data?

How do these estimates compare to those from imputation techniques, including ones that also model a feature distribution?

**Q2** Do higher-quality expectations of a logistic regression classifier result in higher accuracy on test data?

**Q3** Does NaCL retain logistic regression’s higher predictive accuracy over unconstrained naive Bayes?

## Experimental Setup

To demonstrate the generality of our method, we construct a 5-dataset testbed suite that covers assorted configurations [Yann et al., 2009; Xiao et al., 2017b; Blackard and Dean, 1999; Dua and Karra Taniskidou, 2017; Noordewier et al., 1991]; see Table 2.1. The suite ranges from image classification to DNA sequence recognition; from fully balanced labels to  $> 75\%$  of samples belonging to a single class; from continuous to categorical features with up to 40 different values. For datasets with no predefined test set, we construct one by a 80:20 split. As our method assumes binary inputs, we transform categorical features through one-hot encodings and binarize continuous ones based on whether they are 0.05 standard deviation above their respective mean.

Our algorithm takes as input a logistic regression model which we trained using fully observed training data. During prediction time, we make the features go missing uniformly at random based on a set missingness percentage, which corresponds to a missing completely at random (MCAR) mechanism [Little and Rubin, 2014]. We repeat all experiments for 10 (resp. 100) runs on MNIST, Fashion, and CovType (resp. Adult and Splice) and report the average.

## Fidelity to the Original Predictions

The optimal method to deal with missing values would be one that enables the original classifier to act as if no features were missing. In other words, we want the predictions to be affected as little as possible by the missingness. As such, we evaluate the similarity between predictions made with and without missingness, measured by the average cross entropy. The results are reported in Figure 2.2a<sup>7</sup> and Table 2.2; the error bars were too small to be visibly noticeable and were omitted in the plots. In general, our method outperforms all the baselines by a significant margin, demonstrating the superiority of the expected predictions produced by our method.

We also compare NaCL with two imputation methods that consider the feature distribution, namely EM [Dempster et al., 1977] and MICE [Buuren and Groothuis-Oudshoorn, 2010]. EM

---

<sup>7</sup>Max imputation results are dismissed as they are orders of magnitude worse than the rest.

imputation reports the second-to-worst average cross entropies and MICE’s results are very similar to those of mean imputation when 1% of features are missing. Due to the fact that both EM and MICE are excessively time-consuming to run and their imputed values are no better quality than more lightweight alternatives, we do not compare with them in the rest of the experiments. We would like to especially emphasize this comparison; it demonstrates that directly leveraging feature distributions without also considering how the imputed values impact the classifier may lead to unsatisfactory predictions, further justifying the need for solving the expected prediction task and conformant learning. This also concludes our answer to Q1.

### **Classification Accuracy**

Encouraged by the fact that NaCL produces more reliable estimates of the conditional probability of the original logistic regression, we further investigate how much it helps achieve better classification accuracy under different percentages of missing features (i.e., Q2). As suggested by Figure 2.2b and Table 2.3,<sup>8</sup> NaCL consistently outperforms all other methods except on the Adult dataset with 80% of the features missing.

Lastly, to answer Q3 we compare NaCL to a maximum-likelihood naive Bayes model.<sup>9</sup> In all datasets except Splice, logistic regression achieves higher classification accuracy than naive Bayes with fully observed features. NaCL maintains this advantage until 40% of the features go missing, further demonstrating the effectiveness of our method. Note that these four datasets have a large number of samples, which is consistent with the prevailing consensus that discriminative learners are better classifiers given a sufficiently large number of samples [Ng and Jordan, 2002].

---

<sup>8</sup>We report weighted F1 scores as the datasets are unbalanced.

<sup>9</sup>We do not report the full set of results in the table because maximum-likelihood learning of naive Bayes optimizes for a different loss and effectively solves a different task than NaCL and the imputation methods.

## 2.6 Case Study: Sufficient Explanations

In this section we briefly discuss utilizing conformant learning to explain classifications and show some empirical examples as a proof of concept.

On a high level, the task of explaining a particular classification can be thought of as quantifying the “importance” of each feature and choosing a small subset of the most important features as the explanation. Linear models are widely considered easy to interpret, and thus many explanation methods learn a linear model that is closely faithful to the original one, and then use the learned model to assign importance to features [Ribeiro et al., 2016a; Lundberg and Lee, 2017a; Shrikumar et al., 2017]. These methods often assume a black-box setting, and to generate explanations they internally evaluate the predictor on multiple perturbations of the given instance. A caveat is that the perturbed values may have a very low probability on the distribution the classifier was trained on. This can lead to unexpected results as machine learning models typically only guarantee generalization if both train and test data are drawn from the same distribution.

Instead we propose to leverage the feature distribution in producing explanations. To explain a given binary classifier, we consider a small subset of feature observations that is sufficient to get the same classification, in expectation w.r.t. a feature distribution. Next, we formally define our method:

**Definition 4.** (*Support and Opposing Features*) Given  $\mathcal{F}$ ,  $P$ , and  $\mathbf{x}$ , we partition the given feature observations into two sets. The first set consists of the support features that contribute towards the classification of  $\mathcal{F}(\mathbf{x})$ :

$$\mathbf{x}_+ = \begin{cases} \{x \in \mathbf{x} : E_{\mathcal{F},P}(\mathbf{x} \setminus x) \leq \mathcal{F}(\mathbf{x})\} & \text{if } \mathcal{F}(\mathbf{x}) \geq 0.5, \\ \{x \in \mathbf{x} : E_{\mathcal{F},P}(\mathbf{x} \setminus x) > \mathcal{F}(\mathbf{x})\} & \text{otherwise.} \end{cases}$$

The rest are the opposing features that provide evidence against the current classification:  $\mathbf{x}_- = \mathbf{x} \setminus \mathbf{x}_+$ .

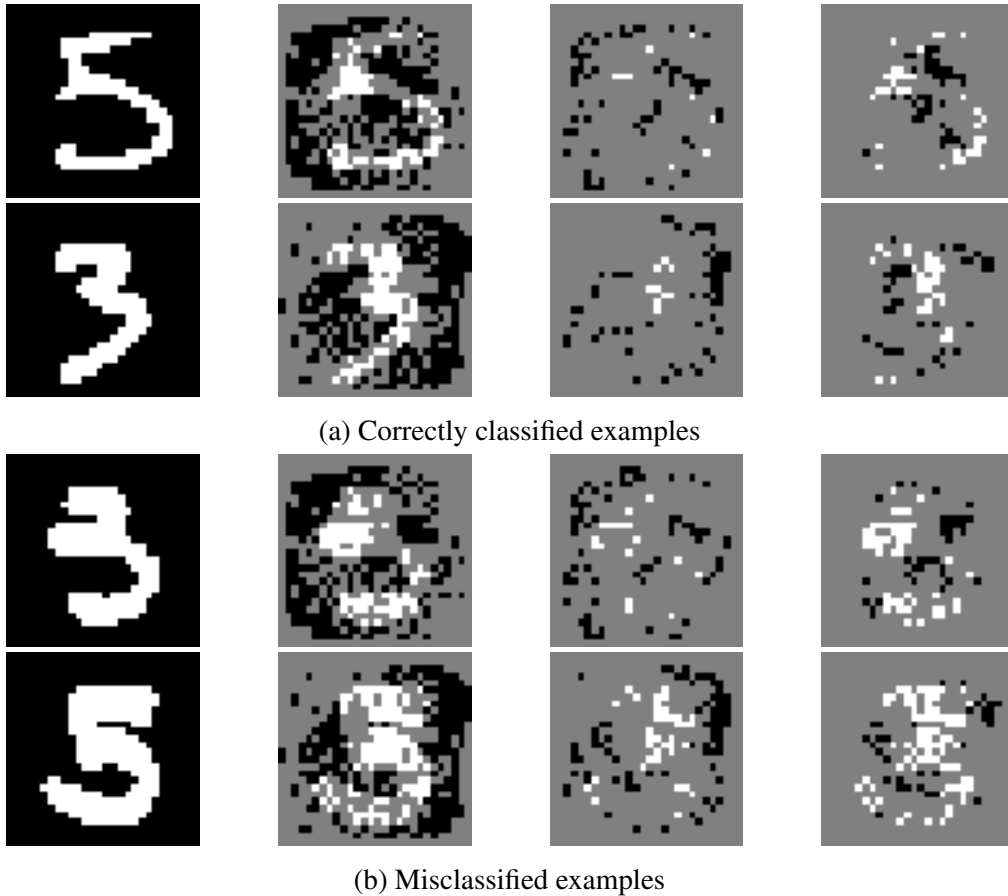


Figure 2.3: Explanations for MNIST classifications. Grey features were not chosen as explanations; white/black are the true color of chosen features. From left to right: 1) all features; 2) support features; 3) top- $k$  support features; 4) sufficient explanation of size  $k$ .

**Definition 5.** Sufficient explanation of  $\mathcal{F}(\mathbf{x})$  with respect to  $P$  is defined as the following:

$$\arg \min_{\mathbf{e} \subseteq \mathbf{x}_+} |\mathbf{e}|$$

$$s.t. \text{sgn}(E_{\mathcal{F}, P}(\mathbf{e}\mathbf{x}_-) - 0.5) = \text{sgn}(\mathcal{F}(\mathbf{x}) - 0.5)$$

Intuitively, this is the smallest set of support features that, in expectation, result in the same classification despite all the evidence to the contrary. In other words, we explain a classification using the strongest evidence towards it.

For a qualitative evaluation, we generate sufficient explanations on instances of a binary logistic

regression task for MNIST digits 5 and 3; see the last column of Figure 2.3. Take the first example in Figure 2.3a: the white pixels selected as sufficient explanation show that the digit should be a 5. Also notice the black pixels in the explanation: they express how the absence of white pixels significantly contributes to the classification, especially in parts of the image where they would be expected for the opposing class. Similarly, the black pixels in the first example in Figure 2.3b look like a 3, and the white pixels in the explanation look like a 5, explaining why this 3 was misclassified as a 5. We further compare our approach to an alternative one that selects a subset of support features based on their logistic regression weights; see the third column of Figure 2.3. It selects features that will cause a large difference in prediction if the value was flipped, as opposed to missing, which is what sufficient explanation considers.

## 2.7 Related Work

There have been many approaches developed to classify with missing values, which can broadly be grouped into two different types. The first one focuses on increasing classifiers' inherent robustness to feature corruption, which includes missingness. A common way to achieve such robustness is to spread the importance weights more evenly among features [Globerson and Roweis, 2006; Dekel and Shamir, 2008; Xia et al., 2017]. One downside of this approach is that the trained classifier may not achieve its best possible performance if no features go missing.

The second one investigates how to impute the missing values. In essence, imputation is a form of reasoning about missing values from observed ones [Sharpe and Solly, 1995; Batista et al., 2002; McKnight et al., 2007]. An iterative process is commonly used during this reasoning process [Buuren and Groothuis-Oudshoorn, 2010]. Some recent works also adapt auto-encoders and GANs for the task [Costa and et al., 2018; Mattei and Frelsen, 2019]. Some of these works can be incorporated into a framework called multiple imputations to reflect and better bound one's uncertainty [Schafer, 1999; Azur et al., 2011; Gondara and Wang, 2018]. These existing methods focus on substituting missing values with those closer to the ground truth, but do not model how

the imputed values interact with the trained classifier. On the other hand, our proposed method explicitly reasons about what the classifier is expected to return.

We are among the first to incorporate feature distributions to generate explanations. Notable recent work along this line proposes to maximize the mutual information between selected features and the class [Chen et al., 2018]. To more explicitly leverage a feature distribution one can explain a classification by a subset of features that maximally affect the classifier output, when its values are substituted by in-fills sampled from the feature distribution conditioned on the rest of the features [Chang et al., 2019]. This contrasts with our method which studies the affect of certain features on a classifier by marginalizing, rather than sampling.

## **Conclusion**

In this chapter, we introduced the expected prediction task, a principled approach to predicting with missing features. It leverages a feature distribution to reason about what a classifier is expected to return if it could observe all features. We then proposed conformant learning to learn joint distributions that conform with and can take expectations of discriminative classifiers. A special instance of it—naive conformant learning—was shown empirically to outperform many existing imputation methods. For future work, we would like to explore conformant learning for other generative-discriminative pairs of models, and extend NaCL to real-valued features.



## CHAPTER 3

### On Tractable Computation of Expected Predictions

Computing *expected predictions* of discriminative models is a fundamental task in machine learning that appears in many interesting applications such as fairness, handling missing values, and data analysis. Unfortunately, computing expectations of a discriminative model with respect to a probability distribution defined by an arbitrary generative model has been proven to be hard in general. In fact, the task is intractable even for simple models such as logistic regression and a naive Bayes distribution. In this paper, we identify a pair of generative and discriminative models that enables tractable computation of expectations, as well as moments of any order, of the latter with respect to the former in case of regression. Specifically, we consider expressive probabilistic circuits with certain structural constraints that support tractable probabilistic inference. Moreover, we exploit the tractable computation of high-order moments to derive an algorithm to approximate the expectations for classification scenarios in which exact computations are intractable. Our framework to compute expected predictions allows for handling of missing data during prediction time in a principled and accurate way and enables reasoning about the behavior of discriminative models. We empirically show our algorithm to consistently outperform standard imputation techniques on a variety of datasets. Finally, we illustrate how our framework can be used for exploratory data analysis.

#### 3.1 Introduction

Learning predictive models like regressors or classifiers from data has become a routine exercise in machine learning nowadays. Nevertheless, making predictions and reasoning about classifier

behavior on unseen data is still a highly challenging task for many real-world applications. This is even more true when data is affected by uncertainty, e.g., in the case of noisy or missing observations.

A principled way to deal with this kind of uncertainty would be to probabilistically reason about the expected outcomes of a predictive model on a particular feature distribution. That is, to compute mathematical expectations of the predictive model w.r.t. a generative model representing the feature distribution. This is a common need that arises in many scenarios including dealing with missing data [Little and Rubin, 2019; Khosravi et al., 2019b], performing feature selection [Yu et al., 2009; Choi et al., 2012, 2017b], handling sensor failure and resource scaling [Galindez Olascoaga et al., 2019], seeking explanations [Ribeiro et al., 2016c; Lundberg and Lee, 2017a; Chang et al., 2019] or determining how “fair” the learned predictor is [Zafar et al., 2015, 2017; Choi et al., 2019].

While dealing with the above expectations is ubiquitous in machine learning, computing the expected predictions of an *arbitrary* discriminative models w.r.t. an *arbitrary* generative model is in general computationally intractable [Khosravi et al., 2019b; Roth, 1996a]. As one would expect, the more expressive these models become, the harder it is to compute the expectations. More interestingly, even resorting to simpler discriminative models like logistic regression does not help reducing the complexity of such a task: computing the first moment of its predictions w.r.t. a naive Bayes model is known to be NP-hard [Khosravi et al., 2019b].

In this work, we introduce a pair of expressive generative and discriminative models for regression, for which it is possible to compute not only expectations, but any moment efficiently. We leverage recent advancements in probabilistic circuit representations. Specifically, we prove that generative and discriminative circuits enable computing the moments in time polynomial in the size of the circuits, when they are subject to some structural constraints which do not hinder their expressiveness.

Moreover, we demonstrate that for classification even the aforementioned structural constraints cannot guarantee computations in tractable time. However, efficiently approximating them becomes doable in polynomial time by leveraging our algorithm for the computations of arbitrary moments.

Lastly, we investigate applications of computing expectations. We first consider the challenging scenario of missing values at test time. There, we empirically demonstrate that computing expectations of a discriminative circuit w.r.t. a generative one is not only a more robust and accurate option than many imputation baselines for regression, but also for classification. In addition, we show how we can leverage this framework for exploratory data analysis to understand behavior of predictive models within different sub-populations.

### 3.2 Expectations and higher order moments of discriminative models

We use uppercase letters for random variables, e.g.,  $X$ , and lowercase letters for their assignments e.g.,  $x$ . Analogously, we denote sets of variables in bold uppercase, e.g.,  $\mathbf{X}$  and their assignments in bold lowercase, e.g.,  $\mathbf{x}$ . The set of all possible values that  $\mathbf{X}$  can take is denoted as  $\mathcal{X}$ .

Let  $p$  be a probability distribution over  $\mathbf{X}$  and  $f : \mathcal{X} \rightarrow \mathbb{R}$  be a discriminative model, e.g., a regressor, that assigns a real value (outcome) to each complete input configuration  $\mathbf{x} \in \mathcal{X}$  (features). The task of computing *the  $k$ -th moment* of  $f$  with respect to the distribution  $p$  is defined as:

$$M_k(f, p) \triangleq \mathbb{E}_{\mathbf{x} \sim p(\mathbf{x})} [(f(\mathbf{x}))^k]. \quad (3.2.1)$$

Computing moments of arbitrary degree  $k$  allows one to probabilistically reason about the outcomes of  $f$ . That is, it provides a description of the distribution of its predictions assuming  $p$  as the data-generating distribution. For instance, we can compute the mean of  $f$  w.r.t.  $p$ :  $\mathbb{E}_p[f] = M_1(f, p)$  or reason about the dispersion (variance) of its outcomes:  $\text{VAR}_p(f) = M_2(f, p) - (M_1(f, p))^2$ .

These computations can be a very useful tool to reason in a principled way about the behavior of  $f$  in the presence of uncertainty, such as making predictions with missing feature values [Khosravi et al., 2019b] or deciding a subset of  $\mathbf{X}$  to observe [Krause and Guestrin, 2009; Yu et al., 2009]. For example, given a partial assignment  $\mathbf{x}^o$  to a subset  $\mathbf{X}^o \subseteq \mathbf{X}$ , the expected prediction of  $f$  over the

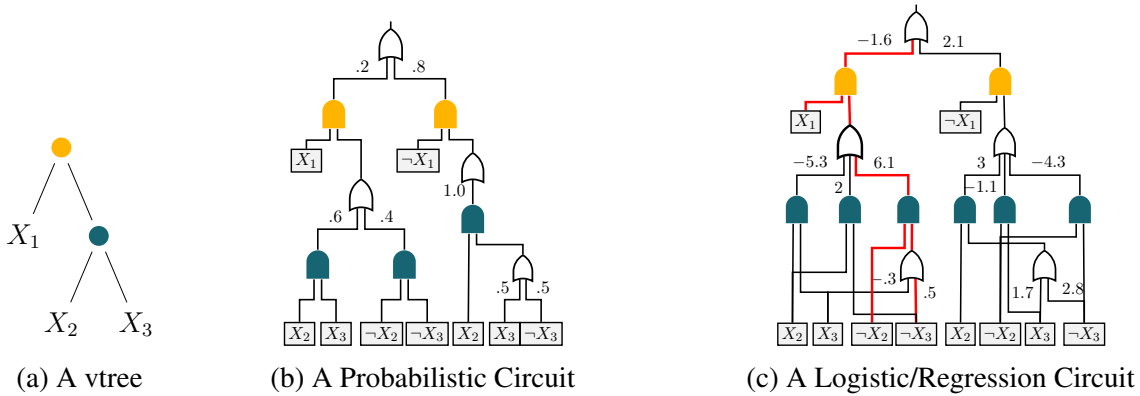


Figure 3.1: A vtree (a) over  $\mathbf{X} = \{X_1, X_2, X_3\}$  and a generative and discriminative circuit pair (b, c) that conform to it. AND gates are colored as the vtree nodes they correspond to (blue and orange). For the discriminative circuit on the right, “hot wires” that form a path from input to output are colored red, for the given input configuration  $\mathbf{x} = (X_1 = 1, X_2 = 0, X_3 = 0)$ .

unobserved variables can be computed as  $\mathbb{E}_{\mathbf{x} \sim p(\mathbf{x}|\mathbf{x}^o)} [f(\mathbf{x})]$ , which is equivalent to  $M_1(f, p(\cdot|\mathbf{x}^o))$ .

Unfortunately, computing arbitrary moments, and even just the expectation, of a discriminative model w.r.t. an arbitrary distribution is, in general, computationally hard. Under the restrictive assumptions that  $p$  fully factorizes, i.e.,  $p(\mathbf{X}) = \prod_i p(X_i)$ , and that  $f$  is a simple linear model of the form  $f(\mathbf{x}) = \sum_i \phi_i x_i$ , computing expectations can be done in linear time. However, the task suddenly becomes NP-hard even for slightly more expressive models, for instance when  $p$  is a naive Bayes distribution and  $f$  is a logistic regression (a generalized linear model with a sigmoid activation function). See Khosravi et al. [2019b] for a detailed discussion.

In Section 3.4, we propose a pair of a generative and discriminative models that are highly expressive and yet still allow for polytime computation of exact moments and expectations of the latter w.r.t. the former. We first review the necessary background material in Section 3.3.

### 3.3 Generative and discriminative circuits

This section introduces the pair of circuit representations we choose as expressive generative and discriminative models. In both cases, we assume the input is discrete. We later establish under

which conditions computing expected predictions becomes tractable.

**Logical circuits** A *logical circuit* [Darwiche and Marquis, 2002; Darwiche, 2003] is a directed acyclic graph representing a logical formula where each node  $n$  encodes a logical sub-formula, denoted as  $[n]$ . Each inner node in the graph is either an AND or an OR gate, and each leaf (input) node encodes a Boolean literal (e.g.,  $X$  or  $\neg X$ ). We denote the set of child nodes of a gate  $n$  as  $\text{ch}(n)$ . An assignment  $\mathbf{x}$  satisfies node  $n$  if it satisfies the logical formula encoded by  $n$ , written  $\mathbf{x} \models [n]$ . Fig. 3.1 depicts some examples of logical circuits. Several syntactic properties of circuits enable efficient logical and probabilistic reasoning over them [Darwiche and Marquis, 2002; Shen et al., 2016]. We now review those properties as they will be pivotal for our efficient computations of expectations and high-order moments in Section 3.4.

**Syntactic Properties** A circuit is said to be *decomposable* if for every AND gate its inputs depend on disjoint sets of variables. For notational simplicity, we will assume decomposable AND gates to have two inputs, denoted  $\mathcal{L}$ (eft) and  $\mathcal{R}$ (ight) children, depending on variables  $\mathbf{X}^{\mathcal{L}}$  and  $\mathbf{X}^{\mathcal{R}}$  respectively. In addition, a circuit satisfies *structured decomposability* if each of its AND gates decomposes according to a *vtree*, a binary tree structure whose leaves are the circuit variables. That is, the  $\mathcal{L}$  (resp.  $\mathcal{R}$ ) child of an AND gate depends on variables that appear on the left (resp. right) branch of its corresponding vtree node. Fig. 3.1 shows a vtree and visually maps its nodes to the AND gates of two example circuits. A circuit is *smooth* if for an OR gate all its children depend on the same set of variables [Shih et al., 2019]. Lastly, a circuit is *deterministic* if, for any input, at most one child of every OR node has a non-zero output. For example, Fig. 3.1c highlights in red the wires that are true, and that form a path from the root to the leaves, given input  $\mathbf{x} = (X_1 = 1, X_2 = 0, X_3 = 0)$ . Note that every OR gate in Fig. 3.1c has at most one hot input wire, because of the determinism property.

**Generative probabilistic circuits** A *probabilistic circuit* (PC) is characterized by its logical circuit structure and parameters  $\theta$  that are assigned to the inputs of each OR gate.

Intuitively, each PC node  $n$  recursively defines a distribution  $p_n$  over a subset of the variables  $\mathbf{X}$  appearing in the sub-circuit rooted at it. More precisely:

$$p_n(\mathbf{x}) = \begin{cases} \mathbb{1}_n(\mathbf{x}) & \text{if } n \text{ is a leaf,} \\ p_{\mathcal{L}}(\mathbf{x}^{\mathcal{L}}) \cdot p_{\mathcal{R}}(\mathbf{x}^{\mathcal{R}}) & \text{if } n \text{ is an AND gate,} \\ \sum_{i \in \text{ch}(n)} \theta_i p_i(\mathbf{x}) & \text{if } n \text{ is an OR gate.} \end{cases} \quad (3.3.1)$$

Here,  $\mathbb{1}_n(\mathbf{x}) \triangleq \mathbb{1}\{\mathbf{x} \models [n]\}$  indicates whether the leaf  $n$  is satisfied by input  $\mathbf{x}$ . Moreover,  $\mathbf{x}^{\mathcal{L}}$  and  $\mathbf{x}^{\mathcal{R}}$  indicate the subsets of configuration  $\mathbf{x}$  restricted to the decomposition defined by an AND gate over its  $\mathcal{L}$  (resp.  $\mathcal{R}$ ) child. As such, an AND gate of a PC represents a factorization over independent sets of variables, whereas an OR gate defines a mixture model. Unless otherwise noted, in this paper we adopt PCs that satisfy *structured decomposability* and *smoothness* as our generative circuit.

PCs allow for the exact computation of the probability of complete and partial configurations (that is, marginalization) in time linear in the size of the circuit. A well-known example of PCs is the probabilistic sentential decision diagram (PSDD) [Kisa et al., 2014].<sup>1</sup> They have been successfully employed as state-of-the-art density estimators not only for unstructured [Liang et al., 2017] but also for structured feature spaces [Choi et al., 2015a; Shen et al., 2017, 2018]. Other types of PCs include sum-product networks (SPNs) and cutset networks, yet those representations are typically decomposable but not structured decomposable [Poon and Domingos, 2011a; Rahman et al., 2014a].

**Discriminative circuits** For the discriminative model  $f$ , we adopt and extend the semantics of logistic circuits (LCs): discriminative circuits recently introduced for classification [Liang and Van den Broeck, 2019a]. An LC is defined by a decomposable, *smooth* and *deterministic* logical circuit with parameters  $\phi$  on inputs to OR gates. Moreover, we will work with LCs that are *structured decomposable*, which is a restriction already supported by their learning algorithms [Liang and Van den Broeck, 2019a]. An LC acts as a classifier on top of a rich set of non-linear features,

---

<sup>1</sup>PSDDs by definition also satisfy determinism, but we do not require this property for computing moments.

extracted by its logical circuit structure. Specifically, an LC assigns an *embedding* representation  $h(\mathbf{x})$  to each input example  $\mathbf{x}$ . Each feature  $h(\mathbf{x})_k$  in the embedding is associated with one input  $k$  of one of the OR gates in the circuit (and thus also with one parameter  $\phi_k$ ). It corresponds to a logical formula that can be readily extracted from the logical circuit structure.

Classification is performed on this new feature representation by applying a sigmoid non-linearity:  $f^{\text{LC}}(\mathbf{x}) \triangleq 1/(1 + e^{-\sum_k \phi_k h(\mathbf{x})_k})$ , and similar to logistic regression it is amenable to convex parameter optimization. Alternatively, one can fully characterize an LC by recursively defining the output of each node  $m$ . We use  $g_m(\mathbf{x})$  to define output of node  $m$  given  $\mathbf{x}$ . It can be computed as:

$$g_m(\mathbf{x}) = \begin{cases} 0 & \text{if } m \text{ is a leaf,} \\ g_{\mathcal{L}}(\mathbf{x}^{\mathcal{L}}) + g_{\mathcal{R}}(\mathbf{x}^{\mathcal{R}}) & \text{if } m \text{ is an AND gate,} \\ \sum_{j \in \text{ch}(m)} \mathbb{1}_j(\mathbf{x})(\phi_j + g_j(\mathbf{x})) & \text{if } m \text{ is an OR gate.} \end{cases} \quad (3.3.2)$$

Again,  $\mathbb{1}_j(\mathbf{x})$  is an indicator for  $\mathbf{x} \models [j]$ , effectively using the determinism property of LCs to select which input to pass through. Then classification is done by applying a sigmoid function to the output of the circuit root  $r$ :  $f^{\text{LC}}(\mathbf{x}) = 1/(1 + e^{-g_r(\mathbf{x})})$ . The increased expressive power of LCs w.r.t. simple linear regressors lies in the rich representations  $h(\mathbf{x})$  they learn, which in turn rely on the underlying circuit structure as a powerful feature extractor [Vergari et al., 2018, 2016].

LCs have been introduced for classification and were shown to outperform larger neural networks [Liang and Van den Broeck, 2019a]. We also leverage them for regression, that is, we are interested in computing the expectations of the output of the root node  $g_r(\mathbf{x})$  w.r.t. a generative model  $p$ . We call an LC when no sigmoid function is applied to  $g_r(\mathbf{x})$  a *regression circuit* (RC). As we will show in the next section, we are able to exactly compute *any moment* of an RC  $g$  w.r.t. an LC  $p$ , that is,  $M_k(g, p)$ , in time polynomial in the size of the circuits, if  $p$  and  $g$  share the same vtree.

### 3.4 Computing expectations and moments for circuit pairs

We now introduce our main result, which leads to efficient algorithms for tractable Expectation and Moment Computation of Circuit pairs ( $EC_2$  and  $MC_2$ ) in which the discriminative model is an RC and the generative model is a PC, and where both circuits are structured decomposable *sharing the same vtree*. Recall that we also assumed all circuits to be smooth, and the RC to be deterministic.

**Theorem 4.** *Let  $n$  and  $m$  be root nodes of a PC and an RC with the same vtree over  $\mathbf{X}$ . Let  $s_n$  and  $s_m$  be their respective number of edges. Then, the  $k^{\text{th}}$  moment of  $g_m$  w.r.t. the distribution encoded by  $p_n$ , that is,  $M_k(g_m, p_n)$ , can be computed exactly in time  $O(k^2 s_n s_m)$ .<sup>2</sup>*

Moreover, this complexity is attained by the  $MC_2$  algorithm, which we describe in the next section. We then investigate how this result can be generalized to arbitrary circuit pairs and how restrictive the structural requirements are. In fact, we demonstrate how computing expectations and moments for circuit pairs *not* sharing a vtree is #P-hard. Furthermore, we address the hardness of computing expectations for an LC w.r.t. a PC—due to the introduction of the sigmoid function over  $g$ —by approximating it through the tractable computation of moments with the  $MC_2$  algorithm.

#### 3.4.1 $EC_2$ : Expectations of regression circuits

Intuitively, the computation of expectations becomes tractable because we can “break it down” to the leaves of the PC and RC, where it reduces to trivial computations. Indeed, the two circuits sharing the same vtree is the property that enables a polynomial time recursive decomposition, because it ensures that pairs of nodes considered by the algorithm depend on exactly the same set of variables.

We will now show how this computation recursively decomposes over pairs of OR and AND gates, starting from the roots of the PC  $p$  and RC  $g$ . We refer the reader to the Appendix for detailed

---

<sup>2</sup>This is a loose upper bound since the algorithm only looks at a small subset of pairs of edges in the circuits. A tighter bound would be  $O(k^2 \sum_v s_v t_v)$  where  $v$  ranges over vtree nodes and  $s_v$  (resp.  $t_v$ ) counts the number of edges going into the nodes of the PC (resp. RC) that can be attributed to the vtree node  $v$ .



proofs of all Propositions and Theorems in this section. Without loss of generality, we assume that the roots of both  $p$  and  $g$  are OR gates, and that circuit nodes alternate between AND and OR gates layerwise.

**Proposition 3.1.** *Let  $n$  and  $m$  be OR gates of a PC and an RC, respectively. Then the expectation of the regressor  $g_m$  w.r.t. distribution  $p_n$  is:*

$$M_1(g_m, p_n) = \sum_{i \in \text{ch}(n)} \theta_i \sum_{j \in \text{ch}(m)} (M_1(\mathbb{1}_j \cdot g_j, p_i) + \phi_j M_1(\mathbb{1}_j, p_i)).$$

The above proposition illustrates how the expectation of an OR gate of an RC w.r.t. an OR gate in the PC is a weighted sum of the expectations of the child nodes. The number of smaller expectations to be computed is quadratic in the number of children. More specifically, one now has to compute expectations of two different functions w.r.t. the children of PC  $n$ .

First,  $M_1(\mathbb{1}_j, p_i)$  is the expectation of the indicator function associated to the  $j$ -th child of  $m$  (see Eq. 3.3.2) w.r.t. the  $i$ -th child node of  $n$ . Intuitively, this translates to the probability of the logical formula  $[j]$  being satisfied according to the distribution encoded by  $p_i$ . Fortunately, this can be computed efficiently, in quadratic time, linear in the size of both circuits as already demonstrated in Choi et al. [2015a].

On the other hand, computing the other expectation term  $M_1(\mathbb{1}_j g_j, p_i)$  requires a novel algorithm tailored to RCs and PCs. We next show how to further decompose this expectation from AND gates to their OR children.

**Proposition 3.2.** *Let  $n$  and  $m$  be AND gates of a PC and an RC, respectively. Let  $n_{\mathcal{L}}$  and  $n_{\mathcal{R}}$  (resp.  $m_{\mathcal{L}}$  and  $m_{\mathcal{R}}$ ) be the left and right children of  $n$  (resp.  $m$ ). Then the expectation of function  $(\mathbb{1}_m \cdot g_m)$  w.r.t. distribution  $p_n$  is:*

$$M_1(\mathbb{1}_m \cdot g_m, p_n) = M_1(\mathbb{1}_{m_{\mathcal{L}}}, p_{n_{\mathcal{L}}}) M_1(g_{m_{\mathcal{R}}}, p_{n_{\mathcal{R}}}) + M_1(\mathbb{1}_{m_{\mathcal{R}}}, p_{n_{\mathcal{R}}}) M_1(g_{m_{\mathcal{L}}}, p_{n_{\mathcal{L}}}).$$

We are again left with the task of computing expectations of the RC node indicator functions,

---

**Algorithm 1**  $EC_2(n, m)$  ▷ Cache recursive calls to achieve polynomial complexity

---

**Require:** A PC node  $n$  and an RC node  $m$

**if**  $m$  is Leaf **then return** 0

**else if**  $n$  is Leaf **then**

**if**  $[n] \models [m_{\mathcal{L}}]$  **then return**  $\phi_{m_{\mathcal{L}}}$

**if**  $[n] \models [m_{\mathcal{R}}]$  **then return**  $\phi_{m_{\mathcal{R}}}$

**else if**  $n, m$  are OR **then return**  $\sum_{i \in \text{ch}(n)} \theta_i \sum_{j \in \text{ch}(m)} (EC_2(i, j) + \phi_j PR(i, j))$

**else if**  $n, m$  are AND **then return**  $PR(n_{\mathcal{L}}, m_{\mathcal{L}}) EC_2(n_{\mathcal{R}}, m_{\mathcal{R}}) + PR(n_{\mathcal{R}}, m_{\mathcal{R}}) EC_2(n_{\mathcal{L}}, m_{\mathcal{L}})$

---

i.e.,  $M_1(\mathbb{1}_{m_{\mathcal{L}}}, p_{n_{\mathcal{L}}})$  and  $M_1(\mathbb{1}_{m_{\mathcal{R}}}, p_{n_{\mathcal{R}}})$ , which can also be done by exploiting the algorithm in Choi et al. [2015a]. Furthermore, note that the other expectation terms ( $M_1(g_{m_{\mathcal{L}}}, p_{n_{\mathcal{L}}})$  and  $M_1(g_{m_{\mathcal{R}}}, p_{n_{\mathcal{R}}})$ ) can readily be computed using Proposition 3.1, since they concern pairs of OR nodes.

We briefly highlight how determinism in the regression circuit plays a crucial role in enabling this computation. In fact, OR gates being deterministic ensures that the otherwise non-decomposable product of indicator functions  $\mathbb{1}_m \cdot \mathbb{1}_k$ , where  $m$  is a parent OR gate of an AND gate  $k$ , results to be equal to  $\mathbb{1}_k$ . We refer the readers to Appendix B.1.3 for a detailed discussion.

Recursively, one is guaranteed to reach pairs of leaf nodes in the RC and PC, for which the respective expectations can be computed in  $\mathcal{O}(1)$  by checking if their associated Boolean indicators agree, and by noting that  $g_m(\mathbf{x}) = 0$  if  $m$  is a leaf (see Eq. 3.3.2). Putting it all together, we obtain the recursive procedure shown in Algorithm 1. Here,  $PR(n, m)$  refer to the algorithm to compute  $M_1(\mathbb{1}_m, p_n)$  in Choi et al. [2015a]. As the algorithm computes expectations in a bottom-up fashion, the intermediate computations can be cached to avoid evaluating the same pair of nodes more than once, and therefore keeping the complexity as stated by our Theorem 4.

### 3.4.2 $MC_2$ : Moments of regression circuits

Our algorithmic solution goes beyond the tractable computation of the sole expectation of an RC. Indeed, any arbitrary order moment of  $g_m$  can be computed w.r.t.  $p_n$ , still in polynomial time. We

call this algorithm  $\text{MC}_2$  and we delineate its main routines with the following Propositions:<sup>3</sup>

**Proposition 3.3.** *Let  $n$  and  $m$  be OR gates of a PC and an RC, respectively. Then the  $k$ -th moment of the regressor  $g_m$  w.r.t. distribution  $p_n$  is:*

$$M_k(g_m, p_n) = \sum_{i \in \text{ch}(n)} \theta_i \sum_{j \in \text{ch}(m)} \sum_{l=0}^k \binom{k}{l} \phi_j^{k-l} M_l(\mathbb{1}_j \cdot g_j, p_i).$$

**Proposition 3.4.** *Let  $n$  and  $m$  be AND gates of a PC and an RC, respectively. Let  $n_{\mathcal{L}}$  and  $n_{\mathcal{R}}$  (resp.  $m_{\mathcal{L}}$  and  $m_{\mathcal{R}}$ ) be the left and right children of  $n$  (resp.  $m$ ). Then the  $k$ -th moment of function  $(\mathbb{1}_m g_m)$  w.r.t. distribution  $p_n$  is:*

$$M_k(\mathbb{1}_m \cdot g_m, p_n) = \sum_{l=0}^k \binom{k}{l} M_l(\mathbb{1}_{m_{\mathcal{L}}} \cdot g_{m_{\mathcal{L}}}, p_{n_{\mathcal{L}}}) M_{k-l}(\mathbb{1}_{m_{\mathcal{R}}} \cdot g_{m_{\mathcal{R}}}, p_{n_{\mathcal{R}}})$$

Analogous to computing simple expectations, by recursively and alternatively applying Propositions 3.3 and 3.4, we arrive at the moments of the leaves at both circuits, while gradually reducing the order  $k$  of the involved moments.

Furthermore, the lower-order moments in Proposition 3.4 that decompose to  $\mathcal{L}$  and  $\mathcal{R}$  children, e.g.,  $M_l(\mathbb{1}_{m_{\mathcal{L}}} \cdot g_{m_{\mathcal{L}}}, p_{n_{\mathcal{L}}})$ , can be computed by noting that they reduce to:

$$M_k(\mathbb{1}_m \cdot g_m, p_n) = \begin{cases} M_1(\mathbb{1}_m, p_n) & \text{if } k = 0, \\ M_k(g_m, p_n) & \text{otherwise.} \end{cases} \quad (3.4.1)$$

Note again that these computations are made possible by the interplay of determinism of  $g$  and shared vtrees between  $p$  and  $g$ . From the former it follows that a sum over OR gate children reduces to a single child value. The latter ensures that the AND gates in  $p$  and  $g$  decompose in the same way, thereby enabling efficient computations.

Given this, a natural question arises: “If we do not require a PC  $p$  and a RC  $g$  to have the same

---

<sup>3</sup>The algorithm  $\text{MC}_2$  can easily be derived from  $\text{EC}_2$  in Algorithm 1, using the equations in this section.

*vtree structure, is computing  $M_k(g, p)$  still tractable?'*. Unfortunately, this is not the case, as we demonstrate in the following theorem.

**Theorem 5.** *Computing any moment of an RC  $g_m$  w.r.t. a PC distribution  $p_n$  where both have arbitrary vtrees is #P-hard.*

At a high level, we can reduce #SAT, a well known #P-complete problem on CNF sentences, to the moment computation problem. Given a choice of different vtrees, we can construct an RC and a PC in time polynomial in the size of the CNF formula such that its #SAT value can be computed using the expectation of the RC w.r.t. the PC. We refer to Appendix B.1.3 for more details.

So far, we have focused our analysis to RCs, the analogous of LCs for regression. One would hope that the efficient computations of  $EC_2$  could be carried on to LCs to compute the expected predictions of classifiers. However, the application of the sigmoid function  $\sigma$  on the regressor  $g$ , even when  $g$  shares the same vtree as  $p$ , makes the problem intractable, as our next Theorem shows.

**Theorem 6.** *Taking the expectation of an LC  $(\sigma \circ g_m)$  w.r.t. a PC distribution  $p_n$  is NP-hard even if  $n$  and  $m$  share the same vtree.*

This follows from a recent result that taking the expectation of a logistic regression w.r.t. a naive Bayes distribution is NP-hard [Khosravi et al., 2019b]; see Appendix B.1.4 for a detailed proof.

### 3.4.3 Approximating expectations of classifiers

Theorem 6 leaves us with no hope of computing exact expected predictions in a tractable way even for pairs of generative PCs and discriminative LCs conforming to the same vtree. Nevertheless, we can leverage the ability to efficiently compute the moments of the RC  $g_m$  to efficiently approximate the expectation of  $\gamma \circ g_m$ , with  $\gamma$  being any differentiable non-linear function, including sigmoid  $\sigma$ . Using a Taylor series approximation around point  $\alpha$  we define the following  $d$ -order approximation:

$$T_d(\gamma \circ g_m, p_n) \triangleq \sum_{k=0}^d \frac{\gamma^{(k)}(\alpha)}{k!} M_k(g_m - \alpha, p_n)$$

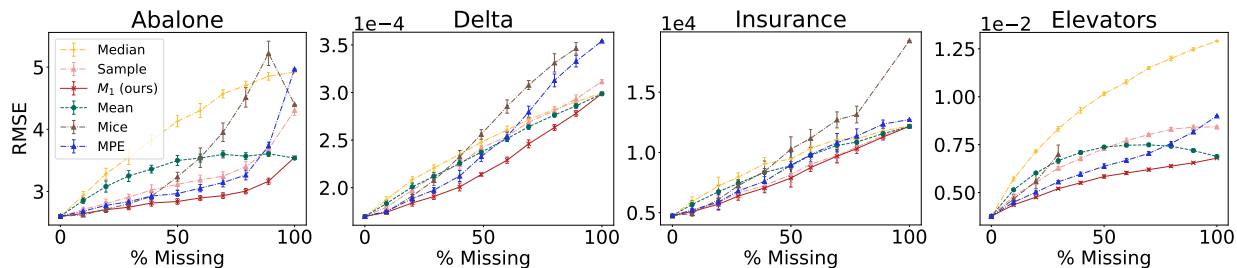


Figure 3.2: Evaluating  $EC_2$  for predictions under different percentages of missing features (x-axis) over four real-world *regression* datasets in terms of the RMSE (y-axis) of the predictions of  $g((\mathbf{x}^m, \mathbf{x}^o))$ . Overall, exactly computing the expected predictions via  $EC_2$  outperforms simple imputation schemes like median and mean as well as more sophisticated ones like MICE [Azur et al., 2011] or computing the MPE configuration with the PC  $p$ . Detailed dataset statistics can be found in Appendix B.2.

See Appendix B.1.5, for a detailed derivation and more intuition behind this approximation.

### 3.5 Expected prediction in action

In this section, we empirically evaluate the usefulness and effectiveness of computing the expected predictions of our discriminative circuits with respect to generative ones.<sup>4</sup> First, we tackle the challenging task of making predictions in the presence of missing values at test time, for both regression and classification.<sup>5</sup> Second, we show how our framework can be used to reasoning about the behavior of predictive models. We employ it in the context of exploratory data analysis, to check for biases in the predictive models, or to search for interesting patterns in the predictions associated with sub-populations in the data distribution.

<sup>4</sup>Our implementation of the algorithm and experiments are available at <https://github.com/UCLA-StarAI/mc2>.

<sup>5</sup>In case of classification, we use the Taylor expansion approximation we discussed in Section 3.4.3.

### 3.5.1 Reasoning with missing values: an application

Traditionally, prediction with missing values has been addressed by imputation, which substitutes missing values with presumably reasonable alternatives such as the mean or median, estimated from training data [Schafer, 1999]. These imputation methods are typically heuristic and model-agnostic [Little and Rubin, 2019]. To overcome this, the notion of expected predictions has recently been proposed in Khosravi et al. [2019b] as a probabilistically principled and model-aware way to deal with missing values. Formally, we want to compute

$$\mathbb{E}_{\mathbf{x}^m \sim p(\mathbf{x}^m | \mathbf{x}^o)} [f(\mathbf{x}^m \mathbf{x}^o)] \quad (3.5.1)$$

where  $\mathbf{x}^m$  (resp.  $\mathbf{x}^o$ ) denotes the configuration of a sample  $\mathbf{x}$  that is missing (resp. observed) at test time. In the case of regression, we can exactly compute Eq. 3.5.1 for a pair of generative and discriminative circuits sharing the same vtree by our proposed algorithm, after observing that

$$\mathbb{E}_{\mathbf{x}^m \sim p(\mathbf{x}^m | \mathbf{x}^o)} [f(\mathbf{x}^m \mathbf{x}^o)] = \frac{1}{p(\mathbf{x}^o)} \mathbb{E}_{\mathbf{x}^m \sim p(\mathbf{x}^m, \mathbf{x}^o)} [f(\mathbf{x}^m \mathbf{x}^o)] \quad (3.5.2)$$

where  $p(\mathbf{x}^m, \mathbf{x}^o)$  is the unnormalized distribution encoded by the generative circuit *configured* for evidence  $\mathbf{x}^o$ . That is, the sub-circuits depending on the variables in  $\mathbf{X}^o$  have been fixed according to the input  $\mathbf{x}^o$ . This transformation, and computing any marginal  $p(\mathbf{x}^o)$ , can be done efficiently in time linear in the size of the PC [Darwiche, 2009a].

To demonstrate the generality of our method, we construct a 6-dataset testing suite, four of which are common regression benchmarks from several domains [Khiari et al., 2018], and the rest are classification on MNIST and FASHION datasets [Yann et al., 2009; Xiao et al., 2017b]. We compare our method with classical imputation techniques such as standard mean and median imputation, and more sophisticated (and computationally intensive) imputation techniques such as multiple imputations by chained equations (MICE) [Azur et al., 2011]. Moreover, we adopt a natural and strong baseline: imputing the missing values by the most probable explanation (MPE)

[Darwiche, 2009a], computed by probabilistic reasoning on the generative circuit  $p$ . Note that the MPE inference acts as an imputation: it returns the mode of the input feature distribution, while  $EC_2$  would convey a more global statistic of the distribution of the outputs of such a predictive model.

To enforce that the discriminative-generative pair of circuits share the same vtree, we first generate a fixed random and balanced vtree and use it to guide the respective parameter and structure learning algorithms of our circuits. In our experiments we adopt PSDDs [Kisa et al., 2014] for the generative circuits. PSDDs are a subset of PCs, since they also satisfy determinism. Although we do not require determinism of generative circuits for moment computation, we use PSDDs due to the availability of their learning algorithms.

On image data, however, we exploit the already learned and publicly available LC structure in [Liang and Van den Broeck, 2019a], which scores 99.4% accuracy on MNIST, being competitive to much larger deep models. We learn a PSDD with the same vtree. For RCs, we adapt the parameter and structure learning of LCs [Liang and Van den Broeck, 2019a], substituting the logistic regression objective with a ridge regression during optimization. For structure learning of both LCs and RCs, we considered up to 100 iterates while monitoring the loss on a held out set. For PSDDs we employ the parameter and structure learning of [Liang et al., 2017] with default parameters and run it up to 1000 iterates until no significant improvement is seen on a held out set.

Figure 3.2 shows our method outperforming other regression baselines. This can be explained by the fact that it computes the exact expectation while other techniques make restrictive assumptions to approximate the expectation. Mean and median imputations effectively assume that the features are independent; MICE<sup>6</sup> assumes a fixed dependence formula between the features; and, as already stated, MPE only considers the highest probability term in the expansion of the expectation.

Additionally, as we see in Figure 3.3, our approximation method for predicted classification, using just the first-order expansion  $T_1(\gamma \circ g_m, p_n)$ , is able to outperform the predictions of the other

---

<sup>6</sup>On the elevator dataset, we reported MICE result only until 30% missing as the imputation method is computationally heavy and required more than 10hr to complete.

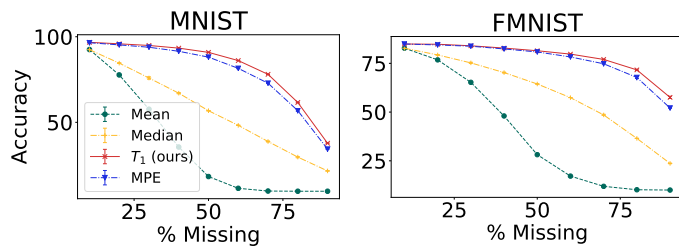


Figure 3.3: Evaluating the first-order Taylor approximation  $T_1(\sigma \circ g_m, p_n)$  of the expected predictions of a *classifier* for missing value imputation for different percentages of missing features (x-axis) in terms of the accuracy (y-axis).

competitors. This suggests that our method is effective in approximating the true expected values.

These experiments agree with the observations from Khosravi et al. [2019b] that, given missing data, probabilistically reasoning about the outcome of a classifier by taking expectations can generally outperform imputation techniques. Our advantage clearly comes from the PSDD learning a better density estimation of the data distribution, instead of having fixed prior assumptions about the features. An additional demonstration of this fact comes from the excellent performance of MPE on both datasets. Again, this can be credited to the PSDD learning a good distribution on the features.

### 3.5.2 Reasoning about predictive models for exploratory data analysis

We now showcase an example of how our framework can be utilized for exploratory data analysis while reasoning about the behavior of a given predictive model. Suppose an insurance company has hired us to analyze both their data and the predictions of their regression model. To simulate this scenario, we use the RC and PC circuits that were learned on the real-world Insurance dataset in the previous section (see Figure 3.2). This dataset lists the yearly health insurance cost of individuals living in the US with features such as age, smoking habits, and location. Our task is to examine the behavior of the predictions, such as whether they are biased by some sensitive attributes or whether there exist interesting patterns across sub-populations of the data.

We might start by asking: “*how different are the insurance costs between smokers and non*



*smokers?*” which can be easily computed as

$$M_1(f, p(\cdot | \text{Smoker})) - M_1(f, p(\cdot | \text{Non Smoker})) = 31,355 - 8,741 = 22,614 \quad (3.5.3)$$

by applying the same conditioning as in Equations 3.5.1 and 3.5.2. We can also ask: “*is the predictive model biased by gender?*” To answer this question, it would be interesting to compute:

$$M_1(f, p(\cdot | \text{Female})) - M_1(f, p(\cdot | \text{Male})) = 14,170 - 13,196 = 974 \quad (3.5.4)$$

As expected, being a smoker affects the health insurance costs much more than being male or female. If it were the opposite, we would conclude that the model may be unfair or misbehaving.

In addition to examining the effect of a single feature, we may study the model in a smaller sub-population, by conditioning the distribution on multiple features. For instance, suppose the insurance company is interested in expanding and as part of their marketing plan wants to know the effect of an individual’s region, e.g., southeast (SE) and southwest (SW), for the sub-population of female (F) smokers (S) with one child (C). By computing the following quantities, we can discover that the difference in their average insurance cost is relevant, but much more relevant is the difference in their standard deviations, indicating a significantly different treatment of this population between regions:

$$\mathbb{E}_{p_{SE}}[f] = M_1(f, p(\cdot | F, S, C, SE)) = 30,974, \text{ STD}_{p_{SE}}[f] = \sqrt{M_2(\cdot) - (M_1(\cdot))^2} = 11,229 \quad (3.5.5)$$

$$\mathbb{E}_{p_{SW}}[f] = M_1(f, p(\cdot | F, S, C, SW)) = 27,250, \text{ STD}_{p_{SW}}[f] = \sqrt{M_2(\cdot) - (M_1(\cdot))^2} = 7,717 \quad (3.5.6)$$

However, one may ask why we do not estimate these values directly from the dataset. The main issue in doing so is that as we condition on more features, fewer if not zero matching samples are present in the data. For example, only 4 and 3 samples match the criterion asked by the last two

queries. Furthermore, it is not uncommon for the data to be unavailable due to sensitivity or privacy concerns, and only the models are available. For instance, two insurance agencies in different regions might want to partner without sharing their data yet.

The expected prediction framework with probabilistic circuits allows us to efficiently compute these queries with interesting applications in explainability and fairness. We leave the more rigorous exploration of their applications for future work.

### **3.6 Related Work**

Using expected prediction to handle missing values was introduced in Khosravi et al. [2019b]; given a logistic regression model, they learned a conforming Naive bayes model and then computed expected prediction only using the learned naive bayes model. In contrast, we are taking the expected prediction using two distinct models. Moreover, probabilistic circuits are much more expressive models. Imputations are a common way to handle missing features and are a well-studied topic. For more detail and a history of the techniques we refer the reader to Buuren [2018]; Little and Rubin [2019].

Probabilistic circuits enable a wide range of tractable operations. Given the two circuits, our expected prediction algorithm operated on the pairs of children of the nodes in the two circuits corresponding to the same vtree node and hence had a quadratic run-time. There are other applications that operate on similar pairs of nodes such as: multiplying the distribution of two PSDDs [Shen et al., 2016], computing the probability of a logical formula [Choi et al., 2015b], and computing KL divergence [Liang and Van den Broeck, 2017].

## **Conclusion**

In this paper we investigated under which model assumptions it is tractable to compute expectations of certain discriminative models. We proved how, for regression, pairing a discriminative circuit

with a generative one sharing the same vtree structure allows to compute not only expectations but also arbitrary high-order moments in poly-time. Furthermore, we characterized when the task is otherwise hard, e.g., for classification, when a non-decomposable, non-linear function is introduced. At the same time, we devised for this scenario an approximate computation that leverages the aforementioned efficient computation of the moments of regressors. Finally, we showcased how the expected prediction framework can help a data analyst to reason about the predictive model's behavior under different sub-populations. This opens up several interesting research venues, from applications like reasoning about missing values, to perform feature selection, to scenarios where exact and approximate computations of expected predictions can be combined.

## CHAPTER 4

# Handling Missing Data in Decision Trees: A Probabilistic Approach

Decision trees are a popular family of models due to their attractive properties such as interpretability and ability to handle heterogeneous data. Concurrently, missing data is a prevalent occurrence that hinders performance of machine learning models. As such, handling missing data in decision trees is a well studied problem. In this paper, we tackle this problem by taking a probabilistic approach. At deployment time, we use tractable density estimators to compute the “expected prediction” of our models. At learning time, we fine-tune parameters of already learned trees by minimizing their “expected prediction loss” w.r.t. our density estimators. We provide brief experiments showcasing effectiveness of our methods compared to few baselines.

### 4.1 Introduction

Decision trees for classification and regression tasks have a long history in ML and AI [Quinlan, 1986; Breiman et al., 1984]. Despite the remarkable successes of deep learning and the enormous attention it attracts, trees and forests are still the preferred off-the-shelf model when in need of robust and interpretable learning on scarce data that is possibly heterogeneous (mixed continuous-discrete) in nature and featuring missing values [Chen and Guestrin, 2016; Devos et al., 2019; Prokhorenkova et al., 2018].

In this work we specifically focus on the last property, noting that while trees are widely regarded as flawlessly handling missing values, *there is no unique way to properly deal with missingness*

*in trees* when it comes to tree induction from data (learning time) or reasoning about partial configurations of the world (deployment time).

Numerous strategies and approaches have been explored in the literature in this regard [Saar-Tsechansky and Provost, 2007; Gavankar and Sawarkar, 2015; Twala et al., 2008]. However, most of these are heuristics in nature [Twala et al., 2008], tailored towards some specific tree induction algorithm [Chen and Guestrin, 2016; Prokhorenkova et al., 2018], or make strong distributional assumptions about the data, such as the feature distribution factorizing completely (e.g., mean, median imputation [Rubin, 1976]) or according to the tree structure [Quinlan, 1993]. As many works have compared the most prominent ones in empirical studies [Batista and Monard, 2003; Saar-Tsechansky and Provost, 2007], there is no clear winner and ultimately, the adoption of a particular strategy in practice boils down to its availability in the ML libraries employed.

In this work, we tackle handling missing data in trees at both learning and deployment time from a principled probabilistic perspective. We propose to decouple tree induction from learning the joint feature distribution, and we leverage tractable density estimators to flexibly and accurately model it. Then we exploit tractable marginal inference to efficiently compute the *expected predictions* Khosravi et al. [2019a] of tree models. In essence, the expected prediction of a tree given a sample with missing values can be thought of as implicitly imputing all possible completions at once, reweighting each complete sample by its probability. In such a way, we can improve the performances of already learned trees, e.g., by XGBoost [Chen and Guestrin, 2016], by making their predictions robust under missing data at deployment time.

Moreover, we show how expected predictions can also be leveraged to deal with *missing data at learning time* by efficiently training trees over the expected version of commonly-used training losses (e.g., MSE). As our preliminary experiments suggest, this probabilistic perspective delivers better performances than common imputation schemes or default ways to deal with missing data in popular decision tree implementations. Lastly, this opens several interesting research venues such as to devise probabilistically principled tree structure induction and robustness against different kinds of missingness mechanisms.

## 4.2 Background

We use uppercase letters ( $X$ ) for random variables (RVs) and lowercase letters ( $x$ ) for their assignments. Analogously, we denote sets of RVs in bold uppercase ( $\mathbf{X}$ ) and their assignments in bold lowercase ( $\mathbf{x}$ ). We denote the set of all possible assignments to  $\mathbf{X}$  as  $\mathcal{X}$ . We denote a *partial* assignment to RVs  $\mathbf{X}^o \subset \mathbf{X}$  as  $\mathbf{x}^o$  and a possible completion to it as  $\mathbf{x}^m$ , that is, an assignment to RVs  $\mathbf{X}^m = \mathbf{X} \setminus \mathbf{X}^o$ .

**Decision trees** Given a set of input RVs  $\mathbf{X}$  (features) and an RV  $Y$  (target) having values in  $\mathcal{Y}$ , a *decision tree*  $f_\Theta$  is a parameterized mapping  $f_\Theta : \mathcal{X} \rightarrow \mathcal{Y}$  characterized by a pair  $(\mathcal{T}, \Theta)$  where  $\mathcal{T}$  is a rooted tree structure and  $\Theta = \{\theta_\ell\}_{\ell \in \text{leaves}(\mathcal{T})}$  is a set of parameters equipped to the *leaves* of  $\mathcal{T}$ . Every non-leaf node  $n$  in  $\mathcal{T}$ , also called a *decision node*, is labeled by an RV  $X_n \in \mathbf{X}$ . For a decision node  $n$  the set of  $k$  outgoing edges  $\{(n, j)\}_{j=1}^k$  partitions  $\mathcal{X}_n$ , the set of values of RV  $X_n$ , into a set of  $k$  disjoint sets  $\mathcal{X}_n^1 \cup \dots \cup \mathcal{X}_n^k = \mathcal{X}_n$ , and defines a set of corresponding decision tests  $\{\llbracket x_n \in \mathcal{X}_n^j \rrbracket\}_{j=1}^k$ . A decision *path*  $\text{path}(\ell)$  is a collection of adjacent edges from the root of  $\mathcal{T}$  to leaf  $\ell$ . Given the above, the mapping encoded in a decision tree  $(\mathcal{T}, \Theta)$  can be written as:

$$f_\Theta(\mathbf{x}) = \sum_{\ell \in \text{leaves}(\mathcal{T})} \theta_\ell \mathcal{I}_\ell(\mathbf{x}) \quad (4.2.1)$$

where  $\mathcal{I}_\ell(\mathbf{x})$  is an indicator function that is equal to 1 if  $\mathbf{x}$  “reaches” leaf  $\ell$  and 0 otherwise; formally,  $\mathcal{I}_\ell(\mathbf{x}) = \prod_{(n,j) \in \text{path}(\ell)} \llbracket x_n \in \mathcal{X}_n^j \rrbracket$ , where  $x_n$  is the assignment for RV  $X_n$  in  $\mathbf{x}$ . The parameters attached to the leaves in  $\mathcal{T}$  here represent a *hard* prediction, i.e.,  $\theta_\ell = y_\ell$  for some value  $y_\ell \in \mathcal{Y}$  associated to leaf  $\ell$ . Our derivations will also hold when  $f_\Theta$  encodes a *soft* predictor, e.g. for  $C$ -class classification,  $f_\Theta : \mathcal{X} \rightarrow [0, 1]^C$ . In that case, we consider a parameter vector  $\boldsymbol{\theta}_\ell$  for each leaf  $\ell$  comprising  $C$  conditional probabilities  $\theta_\ell^i = p(Y = i \mid \mathbf{x})$  for  $i = 1, \dots, C$ .

In the following, we will assume RVs  $\mathbf{X}$  to be discrete. This is to simplify notation and does not hinder generality: our derivations can be easily extended to mixed discrete and continuous RVs by replacing summations to integrations when needed. Note that in the discrete case, a decision node  $n$

labeled by RV  $X_n$  having  $k$  different states, i.e.,  $\mathcal{X}_n = \{1, \dots, k\}$ , will define  $k$  decision tests for one assignment  $x_n$  will be  $\mathbb{I}[x_n = j]$  for  $j = 1, \dots, k$ .

**Decision forests** Single tree models are aggregated in ensembles called *forests* [Breiman, 1996]. One of the most common way to build a forest of  $R$  trees is to put them in a weighted additive ensemble of the form

$$F_{\Theta}(\mathbf{x}) = \sum_{r=1}^R \omega_r f_{\Theta_r}(\mathbf{x}). \quad (4.2.2)$$

This is the case for ensembling techniques like bagging [Breiman, 1996], random forests [Breiman, 2001] and gradient boosting [Friedman, 2001].

**Decision trees for missing data** Several ways have been explored to deal with missing values for decision trees both at training and inference (test) time [Saar-Tsechansky and Provost, 2007]. One of the most common approaches goes under the name of predictive value imputation (PVI) and resorts to replacing missing values before performing inference or tree induction. Among the simplest treatments to missing values in PVI, mean, median and mode imputations are practical and cheap common techniques Rubin [1976]; Breiman [2001]; however, they make strong distributional assumptions like total independence of the feature RVs. More sophisticated (and expensive) PVI techniques cast imputation as prediction from observed features. Among these are multiple imputation with chained equations (MICE) [Buuren and Groothuis-Oudshoorn, 2010] and the use of *surrogate splits* as popularized by CART [Breiman et al., 1984].

Somehow analogous to PVI methods, the missing value treatment done by XGBoost [Chen and Guestrin, 2016] learns to predict which branch to take for a missing feature at inference time by improving some gain criterion on observed data for that feature. While this approach has been proven successful in many real-world scenarios with missing data, it *requires data to be missing at learning time* and it may *overfit* to the missingness pattern observed.

Unlike above imputation schemes, the approach introduced in C4.5 [Quinlan, 1993] replaces imputation with reweighting the prediction associated to one instance by the product of the probabilities of the missing RVs in it. While C4.5 is more distribution aware, these probabilities acting

as weights are only empirical estimates from the training data, and reweighting is limited only to the missing attributes appearing in a path. This assumes that the true distribution over  $\mathbf{X}$  factorizes exactly as the tree structure, which is hardly the case since the tree structure is induced to minimize some predictive loss over  $Y$ .

Several empirical studies showed evidence that there is no clear winner among the aforementioned approaches under different distributional and missingness assumptions [Batista and Monard, 2003; Saar-Tsechansky and Provost, 2007]. In practice, the adoption of a particular strategy is dependent on the specific tree learning or inference algorithm selected, and on the availability of its implemented routines. We introduce in the next section a principled probabilistic and tree-agnostic way of treating missing values at deployment time and extend it to deal with missingness at learning time in Section 4.4.

### 4.3 Expected Predictions of Decision Trees

From a probabilistic perspective, we would like a missing value treatment to be *aware of the full distribution* over RVs  $\mathbf{X}$  without committing to restrictive distributional assumptions. If we have access to the joint distribution  $p(\mathbf{X})$ , then clearly the best way to deal with missing values at inference time would be *to impute all possible completions at once*, weighting them by their probabilities according to  $p(\mathbf{X})$ , thereby generalizing both C4.5 and PVI treatments. This is what the *expected prediction* estimator delivers.

**Definition 6** (Expected prediction). *Given a predictive model  $f_{\Theta} : \mathcal{X} \rightarrow \mathcal{Y}$ , a distribution  $p(\mathbf{X})$  over features  $\mathbf{X}$  and a partial assignment  $\mathbf{x}^o$  for RVs  $\mathbf{X}^o \subset \mathbf{X}$ , the expected prediction of  $f$  w.r.t.  $p$  is:*

$$\mathbb{E}_{\mathbf{x}^m \sim p(\mathbf{X}^m | \mathbf{x}^o)} [f_{\Theta}(\mathbf{x}^o, \mathbf{x}^m)] \quad (4.3.1)$$

where  $\mathbf{X}^m = \mathbf{X} \setminus \mathbf{X}^o$  and  $f_{\Theta}(\mathbf{x}^o, \mathbf{x}^m) = f_{\Theta}(\mathbf{x})$ .

Computing expected predictions is theoretically appealing also because the delivered estimator



is consistent under both MCAR and MAR missingness mechanisms, if  $f$  has been trained on complete data and is Bayes optimal [Josse et al., 2019]. As one would expect, however, computing Equation 4.3.1 exactly for arbitrary pairs of  $f$  and  $p$  is NP-hard [Khosravi et al., 2019c]. Recently, Khosravi et al. [2019a] identified a class of expressive density estimators  $p$  and accurate predictive models  $f$  that allows for polytime computation of the expected predictions of the latter w.r.t. the former. Specifically, probabilistic circuits (PCs) [Choi et al., 2020b] with certain structural restrictions can be used as tractable density estimators to compute the expected predictions exactly for regression and to approximate them for classification, from simple models such as linear and logistic regression to their generalization as circuits [Liang and Van den Broeck, 2019a]. Here we extend those results to compute expected predictions for both classification and regression trees *exactly* and *efficiently* under milder distributional assumptions for  $p$ .

**Proposition 4.1** (Expected predictions for decision trees). *Given a decision tree  $(\mathcal{T}, \Theta)$  encoding  $f_{\Theta}(\mathbf{x})$ , a distribution  $p(\mathbf{X})$ , and a partial assignment  $\mathbf{x}^o$ , the expected prediction of  $f$  w.r.t.  $p$  can be computed as follows:*

$$\mathbb{E}_{\mathbf{x}^m \sim p(\mathbf{X}^m | \mathbf{x}^o)} [f_{\Theta}(\mathbf{x}^o, \mathbf{x}^m)] = \frac{1}{p(\mathbf{x}^o)} \sum_{\ell \in \text{leaves}(\mathcal{T})} \theta_{\ell} \cdot p_{\ell}(\mathbf{x}^o) \quad (4.3.2)$$

where  $p_{\ell}(\mathbf{x}^o) = p(\mathbf{x}^{\text{path}(\ell)}, \mathbf{x}^o)$  and  $\mathbf{x}^{\text{path}(\ell)}$  is the assignment to the RVs in  $\text{path}(\ell)$  that evaluates  $\mathcal{I}_{\ell}(\mathbf{x}') = \prod_{(n,j) \in \text{path}(\ell)} \mathbb{1}[x'_n = j]$  to 1.

We refer to Appendix C.1 for detailed derivations. Note that we can readily extend Equation 4.3.2 to forests of trees (cf. Equation 4.2.2) by linearity of expectations.

As the proposition suggests, we can tractably compute the exact expected predictions of a decision tree if the number of its leaves is polynomial in the input size and we can compute  $p_{\ell}(\mathbf{x}^o)$  in polytime for each leaf  $\ell$ . The first condition generally holds in practice, as trees have low-depth to avoid overfitting, especially in forests, while the second one can be easily satisfied by employing a probabilistic model guaranteeing tractable marginalization, as we need to marginalize over the RVs not in  $\text{path}(\ell)$ . Among suitable candidates are Gaussian distributions and their mixtures for

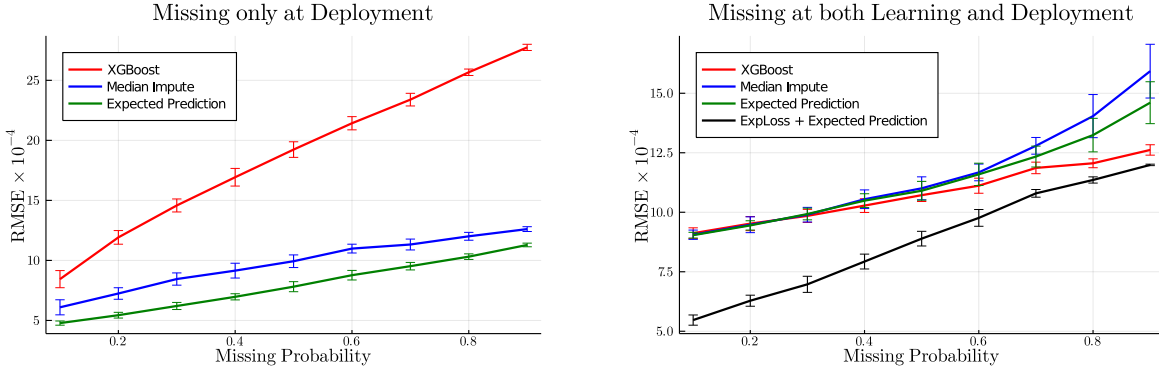


Figure 4.1: Average test RMSE (y-axis, the lower, the better) on the Insurance data for different percentages of missing values (x-axis) when missingness is only at deployment time for a forest of 5 trees (left) or both at learning and deployment time for a single tree learned with XGBoost (right). For each experiment setting, we repeat 10 times and report the average error and their standard deviation.

continuous data, and smooth and decomposable PCs [Choi et al., 2020b] which are deep versions of classical mixture models. We employ PCs in our experiments as they are potentially more expressive than shallow mixtures and can seamlessly model mixed discrete-continuous distributions.

#### 4.4 Expected Parameter Learning of Trees

Expected predictions provide a principled way to deal with missing values at inference time. In the following, we extend them to learn the parameters  $\Theta$  of a predictive model from incomplete data as to minimize the expectation of a certain loss w.r.t. a generative model at hand. We call this learning scenario, *expected loss minimization*.

**Definition 7** (Expected loss minimization). *Given a dataset  $D_{\text{train}}$  over  $\mathcal{X} \times \mathcal{Y}$  containing missing values for RVs  $\mathbf{X}$ , a density estimator  $p_{\Phi}(\mathbf{X})$  trained on  $D_{\text{train}}$  by maximum likelihood, and a per-sample loss function  $l$ , we want to find the set of parameters  $\Theta$  of the predictive model  $f_{\Theta} : \mathcal{X} \rightarrow \mathcal{Y}$  that minimizes the expected loss  $\mathcal{L}(\Theta)$  defined as follows:*

$$\mathcal{L}(\Theta; D_{\text{train}}) = \frac{1}{|D_{\text{train}}|} \sum_{\mathbf{x}^o, y \in D_{\text{train}}} \mathbb{E}_{p_{\Phi}(\mathbf{X}^m | \mathbf{x}^o)} [l(y, f_{\Theta}(\mathbf{x}))]$$

Again, we harness the ability of the density estimator  $p_\Phi$  to accurately model the distribution over RVs  $\mathbf{X}$  and to minimize the loss over  $f_\Theta$  as if it were trained on all possible completions for a partial configuration  $\mathbf{x}^o$ .

For commonly used per-sample losses, the optimal set of parameters for single decision trees can be efficiently and independently computed in closed form. This is for instance the case for the  $L_2$  loss, also known as mean squared error (MSE), defined as  $l_{\text{MSE}}(y, f_\Theta(\mathbf{x})) := (y - f_\Theta(\mathbf{x}))^2$ , which we will use in our experiments.

**Proposition 4.2** (Expected parameters of MSE loss). *Given a decision tree structure  $\mathcal{T}$  and a training set  $D_{\text{train}}$ , the set of parameters  $\Theta = \{\theta_\ell\}_{\ell \in \text{leaves}(\mathcal{T})}$  that minimizes  $\mathcal{L}_{\text{MSE}}$ , the expected prediction loss for MSE, can be found by*

$$\theta_\ell^* = \frac{\sum_{\mathbf{x}^o, y \in D_{\text{train}}} y \cdot p_\ell(\mathbf{x}^o) / p(\mathbf{x}^o)}{\sum_{\mathbf{x}^o, y \in D_{\text{train}}} p_\ell(\mathbf{x}^o) / p(\mathbf{x}^o)}$$

for each leaf  $\ell \in \text{leaves}(\mathcal{T})$ .

The above equation for optimal leaf parameters can be extended to forests of trees where each tree is learned independently, e.g., via bagging. For other scenarios involving forests such as boosting refer to Appendix C.2.

Furthermore, a regularization term may be added to the expected loss to counter overfitting in a regression scenario, e.g., by penalizing the leaf parameter magnitude; this still yields close-form solutions (see Appendix C.1). Next, we will show the effect of tree parameter learning via expected losses on some tree structures that have been induced by popular algorithms such as XGBoost [Chen and Guestrin, 2016]; that is, we will *fine tune* their parameters to optimality given their structures and a tractable density estimator for  $p(\mathbf{X})$ . Investigating how to blend expected loss learning in classical top-down tree induction schemes is an interesting venue we are currently exploring.

## 4.5 Experiments

In this section, we provide preliminary experiments to answer the following questions: **(Q1)** Do expected predictions at deployment time improve predictions over common techniques to deal with missingness for trees? **(Q2)** Does expected loss minimization improve predictions when missing values are present also at learning time?

**Setup** We employ the Insurance dataset, in which we want to predict the yearly medical insurance costs of a patient based their personal data.<sup>1</sup> We consider two scenarios, when data is missing only at deployment time or also at learning time. In both cases, we assume data to be MCAR: given complete data, we make each feature be missing with probability  $\pi \in \{0.1, 0.2, \dots, 0.9\}$  each for 10 independent trials. For each setting, we learn a probabilistic circuit  $p_\Phi$  on the training data as well as a decision tree or forest using the ubiquitous XGBoost.

**Methods** For XGBoost we employ the default parameters. As a simple baseline we use median imputation, estimating the per-feature imputations on the observed portion of the training set. We employ expected predictions over the trees learned by XGBoost for dealing with missing data at deployment time. Lastly, we use expected loss to fine-tune the XGBoost trees and use them for expected predictions at deployment time, which we denote as "ExpLoss + Expected Prediction". We measure performance by the average test root mean squared error (RMSE).

**Missing only at deployment time** Figure 4.1 (left) summarizes our results for **Q1**. Expected prediction outperforms XGBoost and median imputation. Notably, the reason XGBoost performs poorly is that it has not seen any missing values at learning time, in which case the “default” branch it uses in case of missing values always points to the first child. Additionally, median imputation makes the strong assumption that all the features are fully independent, which would explain why expected prediction using PCs does better.

**Missing during both learning and deployment** Figure 4.1 summarizes our results for **Q2**. In this scenario, expected predictions perform on par, up to  $\pi = 0.4$ , with the way XGBoost treats

---

<sup>1</sup>Refer to Appendix C.3 for more information about the dataset.

missing values at deployment time generated from the same missingness mechanism it has been trained on. However, both methods are significantly outperformed by fine-tuning the tree parameters by the expected loss minimization. We leave for future work to investigate what happens with missingness mechanisms that differ at learning and deployment time, or when we adopt other ensembling techniques such as bagging and random forests.

## **Conclusion**

In this work, we introduced expected predictions and expected loss minimization for decision trees and forests as a principled probabilistic way to handle missing data both at training and deployment time, while being agnostic to the tree structure or the way it has been learned. We are currently investigating how to exploit this methodology to extend tree induction schemes under different missing value mechanisms and derive consistency guarantees for the learned estimators.

# CHAPTER 5

## Probabilistic Sufficient Explanations

Understanding the behavior of learned classifiers is an important task, and various black-box explanations, logical reasoning approaches, and model-specific methods have been proposed. In this paper, we introduce *probabilistic sufficient explanations*, which formulate explaining an instance of classification as choosing the “simplest” subset of features such that only observing those features is “sufficient” to explain the classification. That is, sufficient to give us strong probabilistic guarantees that the model will behave similarly when all features are observed under the data distribution. In addition, we leverage tractable probabilistic reasoning tools such as probabilistic circuits and expected predictions to design a scalable algorithm for finding the desired explanations while keeping the guarantees intact. Our experiments demonstrate the effectiveness of our algorithm in finding sufficient explanations, and showcase its advantages compared to Anchors and logical explanations.

### 5.1 Introduction

Machine learning models are becoming ubiquitous, and are being used in critical and sensitive areas such as medicine, loan applications, and risk assessment in courts. Hence, unexpected and faulty behaviors in machine learning models can have significant negative impact on people. As a result, there is much focus on explaining and understanding the behavior of such models. Explainable AI (or XAI) is an active area of research that aims to tackle these issues.

There have been many approaches toward explaining an instance of a classification (called a

local explanation) from different perspectives, including logic-based [Shih et al., 2018; Ignatiev et al., 2019a; Darwiche and Hirth, 2020] or model-agnostic approaches [Ribeiro et al., 2016b; Lundberg and Lee, 2017b; Ribeiro et al., 2018]. Each of these methods have their pros and cons; some focus on scalability and flexibility, and some focus on providing guarantees.

In this work, we strive to *probabilistically* explain an instance of classification. Explanations are partial examples, where we treat the features not in the explanation as missing values. We aim for our explanations to be as simple as possible while providing the following sufficiency guarantee: given only the features in the explanation, with high probability under the data distribution  $\Pr(\mathbf{X})$ , the classifier makes the same prediction as on the full example. Simplicity and sufficiency are often at odds with each other, hence balancing them is a challenging act.

We briefly overview other approaches to local explanations and discuss their pros and cons in the framework of sufficiency and simplicity. Broadly, model-agnostic methods are more scalable and flexible but tend to be not as reliable as logic-based methods in providing sufficiency. On the other hand, logical explanation methods tend to sacrifice simplicity for deterministic guarantees of sufficiency. We then motivate how probabilistic notions of sufficiency can overcome these shortcomings as the foundation for sufficient explanations.

Next, we introduce the probabilistic reasoning tools needed to quantify sufficiency and define sufficient explanations: the *Same-Decision Probability* (SDP) and *Expected Prediction* (EP). We use probabilistic circuits (PCs) [Choi et al., 2020b] to model the probability distribution  $\Pr(\mathbf{X})$  over the features due to their expressivity and tractability in answering complex probabilistic queries. We also explore connections between SDP and expected prediction and show why expected prediction is better suited for providing the probabilistic guarantees for sufficient explanations.

We then formalize the desired properties of sufficient explanations and motivate our choices for sufficiency and simplicity. Sufficiency leads to maximizing the expected prediction, while simplicity leads to constraining the size of explanations as well as choosing the subset that also maximizes the marginal probability. We capture all these with an optimization problem for finding sufficient explanations. Then, we design a scalable algorithm for finding the most likely sufficient explanation

by leveraging tractability of expected prediction.

We provide experiments showcasing the empirical advantage of sufficient explanations and the effectiveness of our search algorithm. Our advantages include: (i) compared to Anchors we get better and more accurate sufficiency guarantees and (ii) our method finds simpler explanations and is more scalable than logical methods. Lastly, we show the tradeoffs between sufficiency and simplicity, and show that slightly reducing the guarantee can lead to simpler explanations with higher likelihood.

## 5.2 Background and Related Work

**Notation.** We use uppercase letters ( $X$ ) for features (random variables) and lowercase letters ( $x$ ) for their value assignments. Analogously, we denote sets of features in bold uppercase ( $\mathbf{X}$ ) and their assignments in bold lowercase ( $\mathbf{x}$ ). We denote the set of all possible assignments to  $\mathbf{X}$  as  $\mathcal{X}$ . Concatenation  $\mathbf{XY}$  denotes the union of disjoint sets. We focus on discrete features unless otherwise noted.

We represent a probabilistic predictor as  $f : \mathcal{X} \rightarrow [0, 1]$  and its decision function as  $\mathcal{C} : \mathcal{X} \rightarrow \{0, 1\}$ , with  $T$  denoting the decision threshold. Hence,  $\mathcal{C}(\mathbf{x}) = \mathbb{I}[f(\mathbf{x}) \geq T]$ . Sometimes we want to directly deal with log-odds instead of probabilities, in which case we use the log-odds predictor  $\mathcal{O} : \mathcal{X} \rightarrow \mathbb{R}$  which is defined as  $\mathcal{O}(\mathbf{x}) = \log \frac{f(\mathbf{x})}{1-f(\mathbf{x})}$ .

**Related Work** Computing explanations of classifiers has been studied from many different perspectives, including logical reasoning, black-box methods, and model-specific approaches. Some try to explain the learned model globally, making it more interpretable [Guidotti et al., 2018; Liang and Van den Broeck, 2019a], while others focus more locally on explaining its prediction for a single instance. Next, we go over some local explanation methods, discuss their pros and cons, and motivate how our framework might solve those issues.



**Model Agnostic Approaches** These methods treat the classifier as a black box. Given an input instance to explain, they perturb the instance in many different ways and evaluate the model on those perturbed instances. Then, they use the results from the perturbations to generate an explanation. Two popular methods under this umbrella are LIME [Ribeiro et al., 2016b] and SHAP [Lundberg and Lee, 2017b]. The main difference between these methods is the heuristics used to obtain perturbed instances and how to analyze the predictions on these local perturbations. Most provide feature attributions, which are real-valued numbers assigned to each feature, to indicate their importance to the decision and in what direction.

A benefit of these methods is that they can be used to explain any model and are generally more flexible and scalable than their alternatives. On the other hand, the downsides are that they can be very sensitive to the choice of local perturbations and might produce over-confident results [Ignatiev et al., 2019b] or be fooled by adversarial methods [Slack et al., 2020; Dimanov et al., 2020]. One of the main reasons for these downsides is that the distribution of the local perturbations tends to be different from the data distribution the classifier was originally trained on. Hence, these approaches do not benefit from the intended generalization guarantees of machine learning models. Moreover, some of the perturbations might be low probability or even impossible inputs, and we might not care as much about their classification outcome.

Additionally, feature attribution methods treat each feature independently and cannot easily capture interactions between bigger subsets of features such as when two features cancel each other's effects. We refer to Camburu et al. [2020] for more such examples and discussion on pros/cons of attribution based explanations.

**Logical Reasoning Approaches** These methods provide explanations with some principled guarantees by leveraging logical reasoning tools. Some approaches use knowledge compilation and tractable Boolean circuits [Shih et al., 2018; Darwiche and Hirth, 2020; Shi et al., 2020], some adopt the framework of abductive reasoning [Ignatiev et al., 2019a,b], and some tackle a specific family of models such as linear models [Marques-Silva et al., 2020], decision trees [Izza et al.,

2020], or tree ensembles [Devos et al., 2020].

The main benefit of these approaches is that they guarantee provably correct explanations, that is they guarantee a certain prediction for all examples described by the explanation. On the other hand, one downside is that they are generally not as scalable (in the number of features) as black-box methods. Another downside is that they need to completely remove the uncertainty from the classifier to be able to use logical tools and therefore become more rigid. In particular, in order to guarantee a certain outcome with absolute certainty, it is often necessary to include many of the features into the explanation, making the explanation more complex.

Sufficient Reasons [Shih et al., 2018; Darwiche and Hirth, 2020] is one example of these methods that selects as an explanation a minimal subset of features guaranteeing that, no matter what is observed for the remaining features, the decision will stay the same. Sufficient reasons, as well as related logical explanations, ensure minimality and deterministic guarantees in the outcome, while as we see later our sufficient explanations ensure probabilistic guarantees instead.

For a recent and more comprehensive comparison of logic-based vs. model-agnostic explanation methods, we refer to Ignatiev et al. [2019b]; Ignatiev [2020].

### 5.3 Motivation and Problem Statement

We will overcome the limitations of both the model-agnostic and logic-based approaches by building local explanation methods that are aware of the distribution over features  $\Pr(\mathbf{X})$ . We assume for now that we have access to an accurate model for the feature distribution, and discuss in Section 5.4.1 how it can be obtained. This distribution will allow us to (i) reason about the classifier’s behavior on realistic input instances and (ii) provide probabilistic guarantees on the veracity of the explanations. We thus take a principled probabilistic approach in explaining an instance of classification.

Intuitively, given an instance  $\mathbf{x}$  and the classifier’s outcome  $\mathcal{C}(\mathbf{x})$ , we would like to choose a subset of features  $\mathbf{z} \subseteq \mathbf{x}$  as the “simplest sufficient explanation.” Firstly, we want it to be sufficient, in that it provides strong probabilistic guarantees about the outcome of the classifier when only

features  $z$  are observed. To this end, we want to maximize some sufficiency metric  $\mathcal{F}(z, \mathbf{x}, f, \text{Pr})$  describing the behavior of our predictor when only the explanation  $z$  is observed. However, naively maximizing this metric can run into the pitfall of building more complex explanations in order to squeeze out tiny improvements in the sufficiency metric. To address this, we have our second metric of simplicity  $S(z, \text{Pr})$ , which we use to define a constraint when maximizing the sufficiency metric. Putting these together, we call a subset  $z$  of an instance  $\mathbf{x}$  a sufficient explanation if it maximizes some sufficiency metric  $\mathcal{F}$  under some simplicity constraint.

With this current definition of sufficient explanations, there is still the possibility that two subsets of a given instance, one a subset of the other, are both simple and have the same degree of sufficiency. This motivates us to further define a minimal sufficient explanation  $z$ , i.e. no subset of  $z$  is also a sufficient explanation. This condition is exactly the meaning of “prime” in prime implicants as they are used in logical explanations. We will see in Section 5.5 how we can use the feature distribution to ensure this minimality property.

## 5.4 Probabilistic Notions of Sufficiency

We next introduce two notions of sufficiency using probabilistic reasoning tools and discuss their pros and cons. Then, we discuss their connections. Finally, we use these tools in Section 5.5 to formalize our definition and formulate finding the desired subsets of features as an optimization problem.

### 5.4.1 Probabilistic Reasoning Tools

Probabilistic reasoning is a hard task in general, so we need to choose our probabilistic model for  $\text{Pr}(\mathbf{X})$  carefully. We choose *probabilistic circuits* (PCs), which given some structural constraints enable tractable and exact computation of probabilistic reasoning queries such as marginals [Choi et al., 2020b]. Moreover, they do so without giving up much expressivity. Another advantage of PCs is that we can learn their structure and parameters from data, allowing us to avoid the exponential

worst-case behavior of other probabilistic models.

The two main probabilistic reasoning tools that we use for our explanations are the *Same Decision Probability* (SDP) [Chen et al., 2012] and *Expected Prediction* (EP) [Khosravi et al., 2019c]. We introduce them next and explore their trade-offs and connections.

First we have SDP [Chen et al., 2012] which, intuitively, gives us the probability that our classifier has the same output as  $\mathcal{C}(\mathbf{x})$  given only some subset of observed features  $\mathbf{z}$ .<sup>1</sup>

**Definition 8** (Same Decision Probability). *Given a classifier  $\mathcal{C}$ , a distribution  $\Pr(\mathbf{X})$  over features, a partition  $\mathbf{ZM}$  of features  $\mathbf{X}$ , and an assignment  $\mathbf{x}$  to  $\mathbf{X}$  (and corresponding  $\mathbf{z} \subseteq \mathbf{x}$ ), the same decision probability (SDP) of  $\mathbf{z}$  w.r.t.  $\mathbf{x}$  is*

$$\text{SDP } \mathbf{xz} = \mathbb{E}_{\mathbf{m} \sim \Pr(\mathbf{M}|\mathbf{z})} [\mathbb{I}[\mathcal{C}(\mathbf{zm}) = \mathcal{C}(\mathbf{x})]].$$

The higher the SDP, the better guarantee we have that the partial example  $\mathbf{z}$  will be classified the same way as the full example  $\mathbf{x}$ . SDP and related notions have been successfully used in applications such as trimming Bayesian network classifiers [Choi et al., 2017a] and robust feature selection [Choi and Van den Broeck, 2018]. Renooij [2018] introduced various theoretical properties and bounds on the SDP.

Other explanation methods that provide sufficiency guarantees can be fit into our framework, using SDP as the sufficiency metric. Notably, Anchors [Ribeiro et al., 2018] can be thought of as an empirical approximation of SDP sufficiency, as they aim to provide sufficiency guarantees based on sampling local perturbations instead of relying on the data distribution. Logical explanations also fit in this framework, as they completely prioritize having deterministic guarantees, i.e. SDP=1, over the need for simplicity. Another example of a method that uses probabilistic sufficiency can be found in Khosravi et al. [2019c], where they explain logistic regression models w.r.t. Naive Bayes data distributions.

---

<sup>1</sup>SDP was originally defined for the classifier being a conditional probability test in distribution  $\Pr$ . Here, we slightly generalize SDP to apply to a distribution  $\Pr$  with a separate classifier  $\mathcal{C}$ .

Expected Prediction is another probabilistic reasoning task that has been shown to be successful in handling missing values in classification and has been studied for different family of models [Khosravi et al., 2019a,c, 2020; Li et al., 2021]. It provides a promising alternative for SDP in explanations. Intuitively, given some partial observation, expected prediction can be thought of as trying all possible ways of imputing the remaining features and computing an average of all subsequent predictions weighted by the probability of each imputation. In many cases, our classifiers directly output a probability and in those cases we can compute expected prediction as follows:

**Definition 9** (Expected Prediction). *Given a probabilistic predictor  $f$ , a distribution  $\Pr(\mathbf{X})$  over features, a partition  $\mathbf{ZM}$  of features  $\mathbf{X}$ , and an assignment  $\mathbf{z}$  to  $\mathbf{Z}$ , the expected prediction of  $f$  on  $\mathbf{z}$  is*

$$EP_f(\mathbf{z}) = \mathbb{E}_{\mathbf{m} \sim \Pr(\mathbf{M}|\mathbf{z})} f(\mathbf{z}\mathbf{m}).$$

For some tasks, we may care more about odds rather than probabilities. In those cases, the predictor usually outputs the log-odds  $\mathcal{O}(\mathbf{x})$ , so we also define expected log-odds:

**Definition 10** (Expected Log-Odds). *Given a log-odds predictor  $\mathcal{O}$ , a distribution  $\Pr(\mathbf{X})$  over features, a partition  $\mathbf{ZM}$  of features  $\mathbf{X}$ , and an assignment  $\mathbf{z}$  to  $\mathbf{Z}$ , the expected log-odds of  $f$  on  $\mathbf{z}$  is*

$$EP_{\mathcal{O}}(\mathbf{z}) = \mathbb{E}_{\mathbf{m} \sim \Pr(\mathbf{M}|\mathbf{z})} \mathcal{O}(\mathbf{z}\mathbf{m}).$$

In this paper, unless otherwise noted, expected prediction denoted as  $EP(\mathbf{z})$  could refer to both cases.

#### 5.4.2 Connections between SDP and EP

The choice between using SDP and expected prediction as a sufficiency metric is important, as it will decide what kind of guarantees our explanations provide and how efficient they are to compute. Here we explore connections between SDP and expected prediction and provide intuition on some advantages of using expected prediction in defining sufficient explanations.

One of the main differences between SDP and expected prediction is their computational complexity. While SDP is an appealing criteria to use for selecting explanations, computing the SDP exactly is computationally hard. In particular, it is  $PP^{PP}$ -hard on Bayesian networks [Choi et al., 2012]. Even on a simple Naive Bayes model for both  $\text{Pr}$  and the classifier, computing SDP is NP-hard [Chen et al., 2013].

On the other hand, expected predictions can be tractably computed for many different pairs of discriminative and generative models. For example, it is known to be tractable for the following cases: (i) logistic regression using a conformant naive Bayes distribution [Khosravi et al., 2019c], (ii) decision trees w.r.t. probabilistic circuits (PCs) [Khosravi et al., 2020], (iii) discriminative circuits w.r.t. PCs [Khosravi et al., 2019a], and (iv) when both the feature distribution and predictor are defined by the same PC distribution  $\text{Pr}$ . In the latter case, the predictor is the conditional probability  $\text{Pr}(c \mid \mathbf{X})$ , and the feature distribution is  $\text{Pr}(\mathbf{X})$ . Then, expected prediction can be reduced to probabilistic marginal inference in PCs which is tractable for decomposable circuits [Choi et al., 2020b].

Another difference comes from how SDP and expected prediction handle the two distinct uncertainties that arise from the feature distribution and the classifier. Although both are aware of the uncertainty from the feature distribution  $\text{Pr}$ , SDP ignores the uncertainty in the classifier since it only deals with the decision function  $\mathcal{C}(\mathbf{x})$ . For example, SDP cannot distinguish between cases when the classifier has low confidence  $f(\mathbf{x}) = T + \epsilon$  (only slightly above decision threshold) and cases when the classifier has high confidence such as  $f(\mathbf{x}) = 0.99$ . In many domains this distinction is vital. For example, doctors care more about the odds of a patient having cancer rather than binary decisions.

For these reasons, we choose to use expected predictions to define and optimize for our explanations. The good news is that optimizing EP also allows us to maximize a lower bound on SDP. The next theorem provides a theoretical lower bound for SDP using expected predictions.

**Theorem 7.** *Given a predictor  $f$  (or  $\mathcal{O}$ ), its thresholded classifier  $\mathcal{C}$ , a positively classified instance*

$\mathbf{x}$  (i.e.  $\mathcal{C}(\mathbf{x}) = 1$ ), a distribution  $\text{Pr}$ , and some subset of the features  $\mathbf{z} \subseteq \mathbf{x}$ , we have:

$$\text{SDP } \mathbf{xz} > \frac{EP(\mathbf{z}) - T}{U(\mathbf{z}) - T}. \quad (5.4.1)$$

where  $EP(\mathbf{z})$  refers to expected prediction, and  $U(\mathbf{z})$  is an upper bound for the predictor after fixing  $\mathbf{z}$ , i.e.  $\forall \mathbf{m} U(\mathbf{z}) \geq f(\mathbf{z}\mathbf{m})$ . Moreover, if  $U(\mathbf{z})$  is a tight bound, then Equation 5.4.1 is also tight.

*Proof.* The proof is included in the appendix.<sup>2</sup> □

Theorem 7 gives us two ways of computing a lower bound for SDP: using a probabilistic predictor  $f$  or a log-odds predictor  $\mathcal{O}$ . Additionally, the theorem can be easily generalized to cases when  $\mathcal{C}(\mathbf{x}) = 0$ . See appendix for a more detailed discussion on which bound is better.

## 5.5 Probabilistic Sufficient Explanations

In this section, we use the aforementioned sufficiency metrics and introduce some desirable constraints to ensure simplicity. Then, we put everything together to formalize sufficient explanations and introduce an optimization problem for finding them. Finally, we introduce a search algorithm for finding sufficient explanations by modifying beam search and leveraging tractability of expected predictions.

To simplify the definitions, we assume without loss of generality that the instance we want to explain is positively classified, i.e.  $\mathcal{C}(\mathbf{x}) = 1$ . In Section 5.4.2, we saw two candidates that we can use for probabilistic sufficiency guarantees. The first was SDP, which focuses on the final decision of the model. The second was expected prediction, which takes the confidence of the model into account. Because of this, along with its computational tractability, we choose to use expected prediction for our sufficiency guarantees. More specifically, we want to maximize the expected prediction of our explanation to ensure our model is confident in its classification. In addition,

---

<sup>2</sup>Available at <http://starai.cs.ucla.edu/papers/WangIJCAI21.pdf>

maximizing the expected prediction also maximizes a lower bound for SDP, so in practice we will still get good SDP guarantees.

Having chosen a suitable sufficiency metric, we must now choose a simplicity constraint. There are multiple candidates to choose from for this as well. For example, we can impose a cardinality constraint or require explanations to have a high enough likelihood. We choose the cardinality constraint as it is easy to decide on a threshold on the explanation size. We can now formalize the notion of sufficient explanations.

**Definition 11** (Sufficient Explanations). *Given a predictor  $f$ , distribution  $\Pr$ , and a positively classified instance  $\mathbf{x}$  (i.e.  $\mathcal{C}(\mathbf{x}) = 1$ ), the set of sufficient explanations for  $\mathbf{x}$  is defined as the solution of the following optimization problem:*

$$\arg \max_{z \subseteq \mathbf{x}} EP(z) \quad s.t. \quad |z| \leq k$$

which we denote as  $SE_k(\mathbf{x})$ .

Having defined sufficient explanations, and following the same line of thought as before, we are now interested in choosing sufficient explanations which are minimal. For logical explanations, any minimal sufficient subset can be chosen. However, we can be more selective in our choices using the feature distribution. A natural choice is to choose the most likely sufficient explanations, as these are the most realistic. By maximizing the marginal probability of the explanation, we also ensure the desired minimality. This is because, for subsets  $z_1$  and  $z_2$  of an instance  $\mathbf{x}$ , if  $z_1 \subseteq z_2$  then  $\Pr(z_1) \geq \Pr(z_2)$ . We thus arrive at the following definition.

**Definition 12** (Most Likely Sufficient Explanations). *Given a predictor  $f$ , a distribution  $\Pr$ , and an instance  $\mathbf{x}$ , the most likely sufficient explanations for  $\mathbf{x}$  are given by:*

$$MLSE_k(\mathbf{x}) = \arg \max_{z \in SE_k(\mathbf{x})} \Pr(z)$$

Next, we devise a search algorithm for finding sufficient explanations. In Section 5.6, we show



that our algorithm works well in practice, ensuring both sufficiency and simplicity.

### **Finding Probabilistic Sufficient Explanations**

To find the most likely sufficient explanations, we use a beam search algorithm to greedily search the space of potential explanations. We do so by keeping track of the top  $b$  candidates (the beam) for explanations for each cardinality based on their expected predictions and marginal probabilities.

In more detail, we begin at level zero with the empty candidate, i.e. no features selected. For each subsequent level, we expand the top  $b$  candidates of the previous level by considering all feature subsets with one more feature than in the previous level, formed by adding a previously unselected feature to each candidate. We then select the new top  $b$  candidates by ranking the expanded states based on EP, breaking ties using Pr, and keeping the top  $b$ . The search stops after level  $k$ . At the same time we keep track of the most likely candidate satisfying the current maximum EP.

Due to the nature of the search, each level of the beam search is highly parallelizable, which can be leveraged to speed up the search. Additionally, by running the algorithm for  $k$  levels, we can also keep track of the best sufficiency guarantee at each level between 1 and  $k$ . Hence, with a little extra book keeping we can keep track of explanations with different sizes, and thus degrees of simplicity, and their corresponding sufficiency guarantees.

Note that the search framework can also be easily adapted for different use cases. For example, we can provide a fixed sufficiency constraint and then find the simplest and most likely explanation with an expected prediction higher than the sufficiency threshold. However, choosing a good threshold for expected prediction is not always straightforward.

## 5.6 Experiments

In this section, we provide several experiments to showcase the effectiveness of our search algorithm in finding sufficient explanations.<sup>3</sup> Additionally, we aim to highlight the advantages of our method in comparison with Anchors and logical explanation methods. More specifically, we would like to answer the following questions:

- Can we find explanations with good sufficiency guarantees? How do they compare with Anchors?
- Does relaxation from logical to probabilistic guarantees lead to much simpler explanations?
- What are the tradeoffs between different sufficiency levels and explanation complexity?

We use the adult and MNIST datasets [Kohavi, 1996; Yann et al., 2009] for our experiments. For each dataset, we model the feature distribution by learning a probabilistic circuit (PC) using the open source Juice library [Dang et al., 2021]. We choose decision forests learned by XGBoost [Chen and Guestrin, 2016] as our classifier, as they are popular and because expected prediction is tractable for forests w.r.t. PCs [Khosravi et al., 2020]. For more detailed information on the datasets, preprocessing steps, learned models, and computing infrastructure, please refer to the appendix.

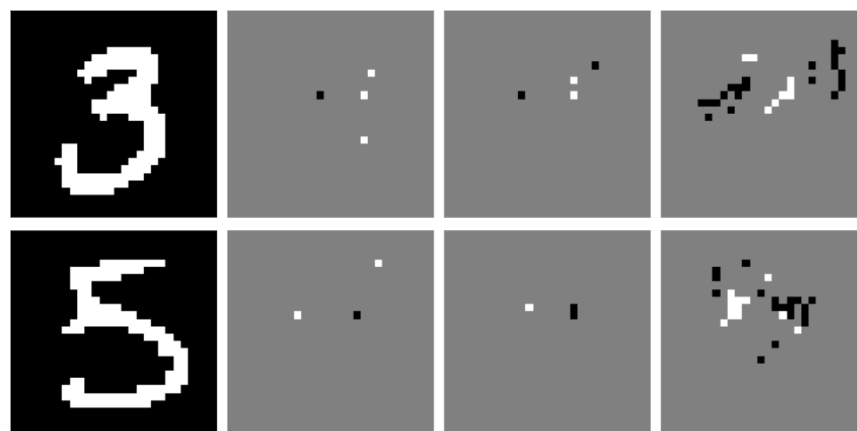
### 5.6.1 Comparison with Anchors

To demonstrate the scalability of our method and showcase some advantages of our method in comparison with Anchors, we ran our algorithm on the binarized MNIST dataset with a binary classification task of distinguishing between 3s and 5s.

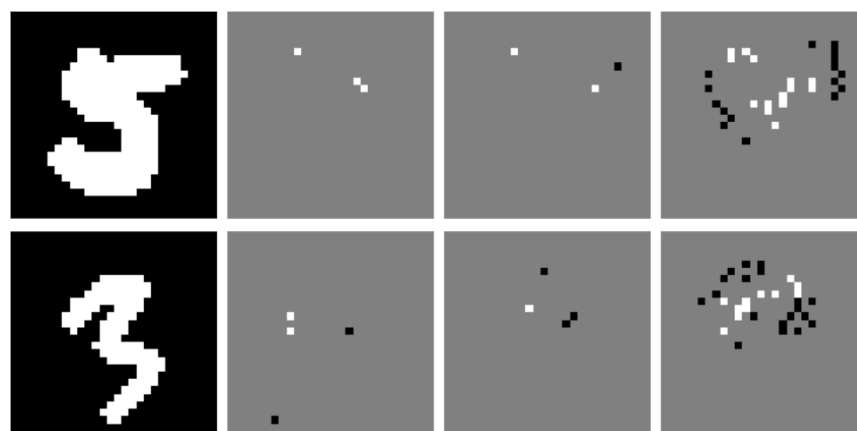
Some images are shown in Figure 5.1 along with a comparison between explanations found using our method and Anchors. For Anchors, we used an SDP (precision) threshold of 0.95, a tolerance ( $\delta$ ) of 0.05, and a beam size of 5. On average it took Anchors 454s to generate explanations.

---

<sup>3</sup>Code at [github.com/UCLA-StarAI/SufficientExplanations](https://github.com/UCLA-StarAI/SufficientExplanations)



(a) Correctly classified examples



(b) Misclassified examples

Figure 5.1: Explanations for selected MNIST images. From left to right: 1) original image; 2) Anchors explanation; 3) our explanation with same number of features 4) our explanation with  $k = 30$ . Gray pixels were not chosen for the explanation. Pixels chosen for the explanation are colored the same color as the original image.

Method	$ \text{EP}_{\mathcal{O}}(\mathbf{z}) $	SDP $\mathbf{xz}$	$\log P(\mathbf{z})$
Anchors	$0.75 \pm 0.37$	$0.66 \pm 0.08$	$-3.29 \pm 0.88$
MLSE <sub>s</sub>	$1.57 \pm 0.29$	$0.86 \pm 0.05$	$-3.05 \pm 0.65$
MLSE <sub>10</sub>	$3.11 \pm 0.23$	$0.99 \pm 0.01$	$-6.98 \pm 1.37$
MLSE <sub>20</sub>	$3.60 \pm 0.15$	$1.00 \pm 0.00$	$-9.90 \pm 2.14$
MLSE <sub>30</sub>	$3.75 \pm 0.13$	$1.00 \pm 0.00$	$-11.77 \pm 2.88$

Table 5.1: Comparison of average expected log-odds, SDP, and marginals between Anchors and MLSE, averaged over 50 random MNIST test images. We take the absolute value of  $\text{EP}_{\mathcal{O}}(\mathbf{z})$  to measure confidence of the explanations (since it could be negative). MLSE<sub>s</sub> sets the cardinality constraint to the same size of the Anchors explanation for each image. The  $\pm$  denotes one standard deviation. The SDP values are approximated.

Our algorithm with the same beam size and cardinality constraint  $k = 30$  took 347s using 16 threads; the sufficient explanations with the same size as Anchors took 45s. See appendix for more details on the run-times.

The last image of each row in Figure 5.1 is the explanation found using our algorithm with cardinality constraint  $k = 30$ . We see that these explanations were able to pick up on certain features we would naturally use to distinguish between 3s and 5s. In particular the chosen pixels were mostly in the upper portion of each image. This makes sense as both 3s and 5s have a similar arch shape in their lower portions, so the upper portion would be more useful for distinguishing between the two. Additionally, the explanations contain not only some white pixels showing an outline of the predicted number, but also some black pixels, where a number of the opposite label may be present. Finally, by looking only at the explanation in the rightmost column of the last two rows we can guess that the classifier will misclassify those examples.

Table 5.1 provides data for the expected log-odds, SDP, and marginal probabilities for generated explanations using Anchors and our method in a few different scenarios. Since SDP is intractable to compute exactly, we estimate it by computing the SDP on 10000 samples drawn from the probabilistic circuit conditioned on the explanation. One advantage of using PCs as our generative model is that drawing conditional samples from  $\text{Pr}(\cdot | \mathbf{z})$  is very fast. For example, generating 10K samples takes 1 second. We see that the Anchors are quite overconfident, giving explanations with

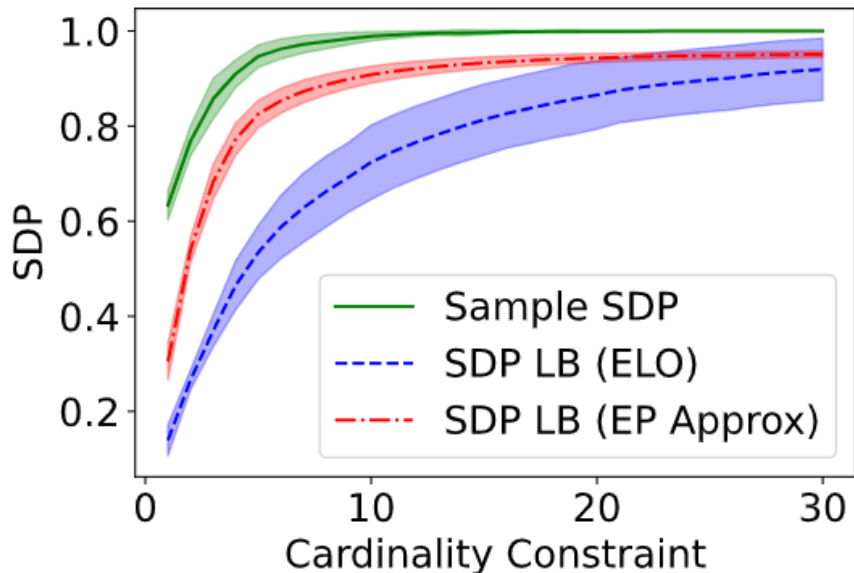


Figure 5.2: For sufficient explanations (sizes 1-30), we plot SDP estimates (green) vs SDP lower bounds calculated based on expected log-odds (blue), and lower bounds based on approximate expected prediction (red), averaged over 50 test images of MNIST. Shaded regions represent one standard deviation.

much lower sample SDPs compared to the desired 0.95. This trend was also observed in Ignatiev et al. [2019b].

We plot the sample SDPs for our explanations, along with lower bounds calculated using Theorem 7, in Figure 5.2. The green line shows that, even when optimizing the EP of our explanations, the SDP still tends to be very high. Moreover, the blue line shows that the simple and efficient SDP lower bound can also provide this guarantee for some of the larger explanations with high EP. The red line is yet another way to estimate the SDP lower bound. See appendix for more details.

### 5.6.2 Comparison with Logical Explanations

We also compared explanations found using our method to logical explanations, i.e. minimal explanations with  $SDP = 1$ . We used the adult dataset, which has much fewer features, for this task in order for the logical explanation computation to become feasible. We removed some features to allow for brute force computation of minimum cardinality logical explanations. We chose to

Method	SDP $xz$	$\log P(z)$	size
Logical	1.0	$-7.12 \pm 2.11$	$4.30 \pm 1.13$
Anchors	$0.98 \pm 0.02$	$-4.27 \pm 2.61$	$2.02 \pm 1.26$
MLSE <sub>s</sub>	$0.97 \pm 0.03$	$-3.85 \pm 2.37$	$2.17 \pm 1.18$
MLSE <sub>1</sub>	$0.88 \pm 0.19$	$-2.23 \pm 1.34$	$1.0 \pm 0.0$
MLSE <sub>2</sub>	$0.95 \pm 0.08$	$-3.88 \pm 1.88$	$2.0 \pm 0.0$
MLSE <sub>3</sub>	$0.98 \pm 0.05$	$-4.77 \pm 2.31$	$2.99 \pm 0.06$
MLSE <sub>4</sub>	$0.99 \pm 0.03$	$-5.63 \pm 2.59$	$3.96 \pm 0.22$

Table 5.2: SDP, marginal probability, and size statistics for explanations found using different methods for the adult dataset.

use brute force because, as far as we know, there is no method for finding logical explanations for our use case. In total we removed 3 features, leaving us with 11 features. From our brute force search on some test examples, we found that logical explanations needed on average 39% of the features in order reach an SDP of one. By using our algorithm of maximizing the expected prediction, we found that in most cases selecting only 18% of the features was already enough to guarantee an SDP of 0.95 on average. More detailed numbers are provided in Table 5.2. We see that the strict requirement imposed by logical explanation methods can lead to selecting more features leading to more complex explanations. We expect this gap between the complexities of logical and probabilistic explanations to widen for datasets with more features or more complex models.

### 5.6.3 Tradeoffs between Sufficiency and Simplicity

Finally, we examine the tradeoff between maximizing sufficiency and explanation simplicity. We compare the expected predictions with marginal likelihoods of explanations. Figure 5.3 shows a scatter plot comparing these two quantities for different explanations generated for 50 MNIST images. As we see, the general trend is that by enforcing less strict sufficiency requirements we can get more likely (and also smaller) explanations. In particular, the trend is very steep around EPs of 0 and 1, meaning that making the guarantees even slightly probabilistic will lead to significant simplification of the explanations, thus validating our probabilistic approach.

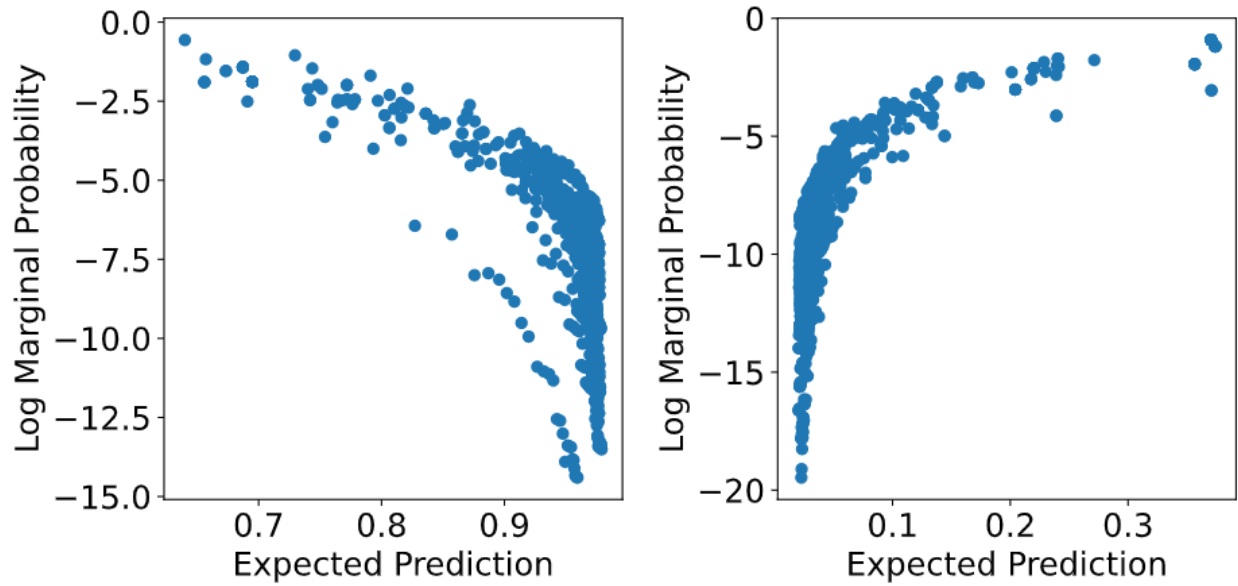


Figure 5.3: Tradeoff between expected prediction and marginal probability for MLSEs. The first plot is for positive label images (5s); the second is for negative label images (3s). Expected predictions were computed using a first order approximation.

## Conclusion

We introduced a new framework for reasoning about the local behavior of classifiers. We formulated the problem as finding simplest and most likely explanations that maximize a probabilistic guarantee, and discussed advantages of our framework compared to model agnostic and logical explanation methods. We provided experiments to validate our claims. We conclude that probabilistic sufficient explanations are a valuable addition to the arsenal of local explanation methods.

## CHAPTER 6

# Why Is This an Outlier? Explaining Outliers by Submodular Optimization of Marginal Distributions.

Detecting outliers is an important task in machine learning, since if left unmitigated they could hinder performance of our models. In this paper, we focus on finding the reason an instance is an outlier, i.e. by finding the subset of outlier features that if ignored the rest of the input is not an outlier anymore. We show how to formulate the problem both as discrete and continuous optimization tasks. Firstly, the discrete search approaches leads to a constrained monotonic submodular optimization task thanks to key properties of marginal distributions. Next, we introduce a continuous relaxation of the search that allows accelerating the search by orders of magnitude. In both continuous and discrete scenarios, we leverage probabilistic circuits (PCs), as our density estimation model to guide the search. The advantage of PCs compared to other density estimation models is that they enable tractable marginal queries for arbitrary subsets which we would need to do repeatedly during search for outliers. We showcase the ability of finding the outlier features in a variety of different outlier benchmarks. Additionally, we show that finding and fixing the outlier features can help in downstream tasks such as classification.

### 6.1 Introduction

In classical machine learning (ML) tasks we rely on the assumption a model is going to be evaluated at test time on data points drawn from the same distribution that generated the data it has been trained on, also called the independently and identically distributed (i.i.d.) assumption [Murphy,



2022]. Therefore *outlier* and *anomalous* points, that is samples that are assumed not to follow the same data distributions [Chandola et al., 2009], can have a considerable impact on performance of ML models at deployment time [Hodge and Austin, 2004]. It becomes clear that detecting these data points, being them natural or adversarially generated [Yuan et al., 2019; Ilyas et al., 2019], early and dealing with them, e.g., by cleaning them [Chu et al., 2016; Liu et al., 2004; Song et al., 2017], is an important task that can positively influence the downstream performances of many ML models: from supervised classifiers [Acuña and Rodriguez, 2004] to unsupervised generative models [Nalisnick et al., 2018].

Classically, these approaches assume that a data point is either an outlier or not (i.e., it is an inlier) and that all features contribute to this dichotomy. As such, when trying to clean it, some methods try to change all its features at once, e.g., for an image data point, all pixels are reset to the most likely frequency values [Song et al., 2017], or more simply they discard the data as a whole. Clearly, this “whole-or-nothing” approach to outlier cleaning can be wasteful as precious data points can be lost.

Furthermore, in many real-world scenarios *a data point is an outlier because only a subset of its features have values that are unlikely according to their marginal distributions*. Consider for example tabular data collecting personal data of patients such as their city and country of birth and their date of birth. The most common ways to create an outlier patient record is by accidentally entering an invalid combination of city and country (e.g. London, Canada instead of London, England) or mistyping the date format (e.g. “day/month/year” vs “month/day/year”) [van den Burg et al., 2019]. Instead of discarding the whole patient record, if one were able to precisely detect these “outlier features”, they could clean them to restore the record . Alternatively, an outlier image can fall outside the natural image distribution because a watermark or an adversarial patch being added to it. To correctly classify the image we would need to i) locate the outlier pixels and ii) marginalize them out, as to still use the remaining inlier ones.

In this paper, we specifically focus on the task of locating the subset of the features that make an outlier an outlier. Specifically, we cast this feature selection problem in a principled probabilistic

framework in which a generative model is iteratively queried for the marginal probability of feature subsets. We propose both discrete and continuous formulations for this optimization problem. The discrete formulations exploit the *supermodularity* and monotonicity of probability measures and distill efficient greedy algorithms as constrained submodular maximization. One discrete variant tries to find the feature subset that maximises the probability of inlier features while the other retrieves the subset of outlier features that minimize the probability. Similarly, we introduce a continuous relaxation of search leveraging convexity of multi-linear extension of submodular functions. The continuous relaxation allows us to accelerate our search for outlier features by orders of magnitude.

Next, we investigate which probabilistic model class can efficiently support our requirement of querying a joint probability distribution multiple times for computing the marginals of different feature subsets. We find this class in framework of probabilistic circuits (PCs) [Vergari et al., 2020; Choi et al., 2020b], tractable computational graphs that encode expressive joint distributions [Peharz et al., 2020a; Liu et al., 2022]. PCs are well-suited for our purposes because, by imposing certain structural constraints over their graphs, we can guarantee to exactly compute the marginal distribution of any feature subset in time linear in the size of the PC [Choi et al., 2020b; Darwiche and Marquis, 2002]. Our proposed methodology can be successfully applied to detect different patterns of outlier features when applied to image data (Section 6.6). Additionally, we show that finding the outlier features and replacing them with more reasonable values could help in downstream tasks such as increasing accuracy of a neural network classifier on corrupted test data. In some cases, the accuracy gains on cleaned data w.r.t. that on corrupted were as high as 30 percent.

## 6.2 Outlier Feature Detection

**Notation.** We use uppercase letters  $X$  for random variables (features) and lowercase letters  $x$  for their value assignments. Similarly, we denote sets of features in bold uppercase  $\mathbf{X}$  and their assignments in bold lowercase  $\mathbf{x}$ . In certain cases, we also use uppercase letters, for example  $S$ , to

denote sets. We use subscripts to denote subset of features (and their assignment), for example  $\mathbf{x}_{in}$  ( $\mathbf{x}_{out}$ ) corresponds to subset of inlier (outlier) features for assignment  $\mathbf{x}$ .

**Problem statement.** In most ML scenarios we make the assumption that our data points  $\mathbf{x}$  are i.i.d. drawn from a distribution  $p(\mathbf{X})$ . With this interpretation, an outlier  $\mathbf{x}'$  is intuitively an instance that is not drawn from  $p(\mathbf{X})$ . This means that in expectation  $\mathbf{x}'$  has zero or very low probability under  $p(\mathbf{X})$ . This is why one popular solution to mark outliers is to select all those instances  $\mathbf{x}'$  whose probability, according to a parameterized probabilistic model  $p_\theta(\mathbf{X}) \approx p(\mathbf{X})$ , falls under a certain threshold  $t$  [Bishop, 1994], i.e.,  $p_\theta(\mathbf{x}') \leq t$ . While other criteria to deem an instance an outlier exist [Thudumu et al., 2020; Wang et al., 2019; Boukerche et al., 2020], classically all of them assume that the whole  $\mathbf{x}'$  is either an outlier or inlier (e.g.,  $p_\theta(\mathbf{x}') > t$ ), and they do not care about the reason *why* it is as such.

In this paper, we go one step further and want to find an *explanation* why  $\mathbf{x}'$  is an outlier. We assume that a hidden corruption process altered a subset of the features of  $\mathbf{x}$ , denoted as  $\mathbf{x}_{out}$  and turned them into *outlier features*, that is features with low-probability under  $p_\theta$ . Furthermore we assume that the rest of the features  $\mathbf{x}_{in} = \mathbf{x} \setminus \mathbf{x}_{out}$  are inlier features, that is the marginal  $\mathbf{x}_{in}$  has high probability under  $p_\theta$ . From our assumptions it follows that if we are able to precisely detect  $\mathbf{x}_{in}$  or equivalently  $\mathbf{x}_{out}$  we can either try to “fix” an outlier instance by replacing  $\mathbf{x}_{out}$  with some high-probability values for the corresponding features, or simply marginalize them out if the ML model that operates on the downstream task is able to deal with missing values. For example, if the classifier can marginalize unobserved values or it is possible to compute its expectation w.r.t. the distribution  $p_\theta(\mathbf{x}_{out} \mid \mathbf{x}_{in})$  [Khosravi et al., 2019a, 2020]).

Ideally, given a data point  $\mathbf{x}$  that is likely an outlier, we would like to find a *minimal subset* of outlier features  $\mathbf{x}_{out}$ , without which,  $\mathbf{x}_{in}$  is an inlier. We relax the constraint of finding a minimal subset into the problem of finding a subset of features with bounded cardinality  $k$ . This can be formalized as follows.

**Definition 13** (Minimizing the probability of outlier features). *Let  $\mathbf{x}$  be an outlier according to*

model  $p_\theta$  and  $k$  be the maximum number of outlier features  $\mathbf{x}_{out}$  that turn  $\mathbf{x}$  into an outlier. Then  $\mathbf{x}_{out}$  can be found by optimizing the following cardinality constraint optimization problem:

$$\mathbf{x}_{out} = \arg \max_{|S| \leq k} 1 - p_\theta(\mathbf{x}_S). \quad (6.2.1)$$

The intuition behind Equation 6.2.1 is that in order to minimize the probability of an outlier, we want to maximise the complement of its probability, a quantity sometimes called the probabilistic margin [Krishnakumar, 2007]. However, Definition 13 does not tell us if and under which conditions Equation 6.2.1 can be solved efficiently. The next section answers affirmatively, showing that the cardinality constrained problem in Equation 6.2.1 can be efficiently approximated in  $O(kn)$  time, where  $k$  is the subset cardinality,  $n$  is the maximum number of features. This can be achieved by noting that the probability measure  $p_\theta$  is a *supermodular function* when defined over discrete features  $\mathbf{X}$ .

The above computational result, however, assumes that we can efficiently and exactly compute the marginals  $p_\theta(\mathbf{x}_S)$  for all possible feature subsets  $S \subseteq \{1, \dots, n\}$ . We discuss a class of efficient and expressive tractable computational graphs, called smooth and decomposable probabilistic circuits [Vergari et al., 2020; Choi et al., 2020b], that exactly enable us to compute arbitrary marginals exactly in time linear in their size, in Section 6.4.

### 6.3 Submodular Optimization for Efficient Outlier Feature Detection

Submodular and supermodular functions naturally occur in many real world applications and hence their properties are a well studied topic, specially under the light that they enjoy efficient approximation algorithms with guarantees under different optimization settings [Liu et al., 2020; Krause and Golovin, 2014]. In this section, we give a quick background of submodular and supermodular optimization relevant to our task.

Given a set of elements  $\Omega$ , the *set function*  $f : 2^\Omega \rightarrow \mathbb{R}$  assigns a utility value to each subset

of  $\Omega$ . Intuitively, a function is submodular if the marginal benefit of adding an specific element to the current set decreases as the set grows. Similarly, a set function is supermodular if its negation is submodular. Additionally, a submodular function is monotone if adding new elements always increases the value. The next definitions ground these concepts in a more formal way.

**Definition 14** (Submodularity). *Let  $\Omega$  be a set, then the function  $f : 2^\Omega \rightarrow \mathbb{R}$  is submodular iff for all  $A, B$  such that  $A \subset B \subseteq \Omega$ , for all  $i \in \Omega \setminus B$ , we have*

$$f(A \cup \{i\}) - f(A) \geq f(B \cup \{i\}) - f(B).$$

**Definition 15** (Supermodularity). *A set function  $f : 2^\Omega \rightarrow \mathbb{R}$  is supermodular iff  $-f$  is submodular.*

**Definition 16** (Monotonicity). *A set function  $f : 2^\Omega \rightarrow \mathbb{R}$  is monotone iff for all  $A, B$  such that  $A \subseteq B \subseteq \Omega$  then  $f(A) \leq f(B)$ .*

Given a monotone submodular set function  $f$ , minimizing and maximizing  $f(S)$  is trivial if there are no constraints, since  $f(\Omega)$  is the maximum and the minimum is  $f(\{i\})$  for some  $i \in \Omega$ .

In the presence of cardinality constraints, such as our case in Equation 6.2.1, in order to maximise a monotone submodular function  $f$  one can exploit a greedy algorithm that provides a  $1 - \frac{1}{e}$  approximation factor to the following optimization problem [Williamson, 2019]:

$$\arg \max_S f(S) \quad \text{s.t.} \quad |S| \leq k. \tag{6.3.1}$$

The algorithm starts with an empty sets  $S = \emptyset$ , and at each step chooses the “best” element to add to  $S$  until  $|S| = k$ , hence we need  $O(|\Omega| \cdot k)$  evaluations of  $f(S)$ . There are many other variants of the greedy algorithm such as lazy greedy [Leskovec et al., 2007], or even lazier [Mirzasoleiman et al., 2014] that provide similar approximate guarantees in expectation but with much less evaluations of  $f(S)$ . However, as we see in Section 6.4, the lazier approaches might not be easily adaptable to our use case to provide better runtime performance than the basic greedy algorithm. This is mostly due to the ability of batching multiple independent calls to  $f(S)$  in our

setting. We leave adaption of lazier approaches for future work.

To show that Equation 6.2.1 enjoys these approximation guarantees, we need to demonstrate that  $1 - p_\theta(\mathbf{x}_S)$  is a monotone submodular function, or alternatively that  $p_\theta(\mathbf{x}_S)$  is a monotone supermodular function, according to definition Definition 15.

We do this in the next section, where we will make use of two more nice properties of submodular functions: i) complement of a submodular function is still submodular ii) if we apply a bijection to a submodular function we will still have a submodular function. More formally:

**Lemma 8 (Complement).** *If  $f : 2^\Omega \rightarrow \mathbb{R}$  is a submodular set function, then the complement function  $g(A) = f(\Omega \setminus A)$  is also submodular.*

**Lemma 9 (Bijection).** *Let  $f : 2^\Omega \rightarrow \mathbb{R}$  be a submodular function, and  $g : Y \rightarrow \Omega$  be a bijection. The set function  $h : 2^Y \rightarrow \mathbb{R} : A \mapsto f(g(A))$  is submodular.*

### 6.3.1 Supermodularity and Monotonicity of Marginal Probabilities

In this section, we review the background for two desired properties of marginals of probability distributions: supermodularity and monotonicity. These properties help us formulate our subset selection problem as a monotone sub-modular optimization task which enjoys fast greedy algorithms with approximation guarantees.

Given a joint distribution  $p(\mathbf{x})$  on all features, we can marginalize a subset of features by summing them out. For example, if we partition the features into two subsets  $S, S'$  then we can marginalize  $S'$  as follows <sup>1</sup>:

$$p(\mathbf{x}_S) = \int_{\text{val}(\mathbf{X}_{S'})} p(\mathbf{x}_S, \mathbf{x}_{S'}) d\mathbf{X}_{S'}.$$

Where  $\text{val}(\mathbf{X}_{S'})$  is set of potential values for  $\mathbf{X}_{S'}$ . Now given an instance  $\mathbf{x}$ , we can treat the marginal probability function as a set function by defining  $f(S) = p(\mathbf{x}_S)$ . If  $p$  is a probability

---

<sup>1</sup>In case of discrete features, the integrals turn into summations.

measure, then we can show that  $f$  will be supermodular:

**Lemma 10** (Supermodularity of probability measures). *Given an assignment  $\mathbf{x}$  and a probability measure  $p$ , the marginal probability function  $f : 2^{\mathbf{X}} \rightarrow \mathbb{R}$  such that  $f(S) = p(\mathbf{x}_S)$  is supermodular.*

**Lemma 11** (Monotonicity of probability measures). *Let  $A, B$  be two events such that  $A \subseteq B$ , and  $p$  be a probability measure, then  $p(A) \leq p(B)$ .*

Note that if we have discrete features, we can simply use the probability mass function as our probability measure. However, if we have continuous features the density function is not a probability measure. In that case, one simple probability measure could be the cumulative distribution function. In our experiments we consider pixels as discrete features and therefore we turn the integral in Section 6.3.1 into a summation.

### 6.3.2 Other greedy approximations (without guarantees)

In the following, we explore an alternative formulation of our outlier feature selection problem. We start from the intuition that a subset of features is an outlier given the specific values of the remaining features.

**Definition 17** (Minimizing marginal conditional probability of outlier features). *Let  $\mathbf{x}$  be an outlier according to model  $p_\theta$  and  $k$  be the maximum number of outlier features  $\mathbf{x}_{out}$  that turn  $\mathbf{x}$  into an outlier given the features  $\mathbf{x}_{in} = \mathbf{x} \setminus \mathbf{x}_{out}$ . They can be found by minimizing the following cardinality constraint optimization problem:*

$$\mathbf{x}_{out} = \arg \min_{|S| \leq k} p_\theta(\mathbf{x}_S \mid \mathbf{x} \setminus \mathbf{x}_S). \quad (6.3.2)$$

Interestingly, solving the minimization problem of Equation 6.3.2 can be done by finding the set of features that *maximises* the probability of a data point of being an inliner.

**Proposition 6.1.** *Let  $\mathbf{x}$  be an outlier according to model  $p_\theta$  and  $k$  be the maximum number of*

outlier features  $\mathbf{x}_{out}$  that turn  $\mathbf{x}$  into an outlier. Solving the cardinality constraint minimization problem of Equation 6.3.2 is equivalent to solve the following maximization problem:

$$\mathbf{x}_{out} = \arg \max_{|S| \leq k} p(\mathbf{x} \setminus \mathbf{x}_S) \quad (6.3.3)$$

*Proof.* First we can use Bayes rule to rewrite the conditional probability  $p(\mathbf{x}_S \mid \mathbf{x} \setminus \mathbf{x}_S)$ , thus obtaining

$$\mathbf{x}_{out} = \arg \min_{|S| \leq k} p(\mathbf{x}_S, \mathbf{x} \setminus \mathbf{x}_S) / p(\mathbf{x} \setminus \mathbf{x}_S) = \arg \min_{|S| \leq k} p(\mathbf{x}) / p(\mathbf{x} \setminus \mathbf{x}_S).$$

Then we can invert the fraction and turn the minimization into a maximization problem. Finally, by noting that  $p(\mathbf{x})$  is constant in our setting, we retrieve Equation 6.3.3:

$$\mathbf{x}_{out} = \arg \min_{|S| \leq k} p(\mathbf{x}) / p(\mathbf{x} \setminus \mathbf{x}_S) = \arg \max_{|S| \leq k} p(\mathbf{x} \setminus \mathbf{x}_S) / p(\mathbf{x}) = \arg \max_{|S| \leq k} p(\mathbf{x} \setminus \mathbf{x}_S).$$

□

Unfortunately, this problem does not enjoy the  $1 - 1/\epsilon$  greedy approximation as Equation 6.2.1, because  $p(\mathbf{x} \setminus \mathbf{x}_S)$  is supermodular and not submodular, according to Lemma 10 and Lemma 8. Nevertheless, we use the greedy algorithm as a practical heuristic that produces reasonable results in practice, as our experiments show in Section 6.6.

## 6.4 Probabilistic models for tractable marginals of arbitrary subsets

Our algorithms are model-agnostic: one can model  $p_\theta$  with any family of generative models that can provide fast and accurate approximation of marginal probabilities for arbitrary subset of features. For example, one can use sampling to approximate marginals of Variational Auto Encoder (VAEs), and some variants of VAEs such as HiVAE even provide a fast approximate marginals with one feed-forward evaluation [Nazabal et al., 2018]. However, the quality of the solutions of the greedy



approximation algorithms can quickly degrade if the internal routine to compute marginals delivers unreliable approximations. In the following we focus on a class of deep generative models that can be as expressive as VAEs [Liu et al., 2022] but at the same time guarantee the *exact* computation of any marginals in linear time. These are probabilistic circuits (PCs).

PCs [Choi et al., 2020a] are a family of models that unify representation of many families of tractable probabilistic models such as arithmetic circuits [Darwiche, 2003], probabilistic sentential decision diagrams [Kisa et al., 2014], and sum product networks [Poon and Domingos, 2011b].

**Definition 18** (Probabilistic circuits). *A probabilistic circuit  $p_\theta$  over variables  $\mathbf{X}$  is a parameterized computational graph that defines a joint probability distribution over  $\mathbf{X}$ . Its structure consists of three types of units: sum, product and distribution units. Each unit  $n$  defines a joint probability distribution over its scope, i.e., the set of variables it is defined on and recursively defined as:  $\phi(n) = \bigcup_{c \in \text{ch}(n)} \phi(c)$ , where  $\text{ch}(n)$  denote the set of children, or inputs, of an inner unit  $n$ . The output of unit  $n$  in the circuit on input  $\mathbf{x}$  is:*

$$p_n(\mathbf{x}) = \begin{cases} g_n(\mathbf{x}) & \text{if } n \text{ is a distribution unit} \\ \prod_{c \in \text{ch}(n)} p_c(\mathbf{x}) & \text{if } n \text{ is a product unit} \\ \sum_{c \in \text{ch}(n)} \theta_{n,c} \cdot p_c(\mathbf{x}) & \text{if } n \text{ is a sum unit} \end{cases}$$

where  $\theta_{n,c} > 0$ , is the parameter associated to the edge  $(n, c)$ . For an input distribution unit  $n$ ,  $g_n(\mathbf{x})$  is usually a simple distribution defined on one of the features, for example Categorical or Gaussian. Finally, the likelihood  $p(\mathbf{x})$  defined by the circuit is the output of its root unit.

The remarkable property of PCs is the ability to compute *many complex functions* of  $p$  in *polytime* if their computational graphs satisfy certain structural properties. As we are interested in efficient and exact marginals, we will require only two structural properties: *smoothness* and *decomposability*.

**Definition 19** (Smoothness & Decomposability). *A circuit is smooth if for every sum unit  $n$ , its inputs depend on the same variables:  $\forall c_1, c_2 \in \text{ch}(n), \phi(c_1) = \phi(c_2)$ . It is decomposable if the inputs of*

every product unit  $n$  depend on disjoint sets of variables:  $\text{ch}(n) = \{c_1, c_2\}, \phi(c_1) \cap \phi(c_2) = \emptyset$ .

**Proposition 6.2** (Tractable marginalization, [Choi et al., 2020b]). *Let  $p$  be a smooth and decomposable circuit over  $\mathbf{X}$  with input distributions that can be tractably marginalized. Then for any variables  $\mathbf{Y} \subseteq \mathbf{X}$  and their assignment  $\mathbf{y}$ , the marginal  $\int_{\mathbf{z} \in \text{val}(\mathbf{Z})} p(\mathbf{y}, \mathbf{z}) d\mathbf{Z}$  can be computed exactly in  $\Theta(|p|)$  time, where  $\mathbf{Z}$  denotes  $\mathbf{X} \setminus \mathbf{Y}$ .*

Smooth and decomposable PCs are both expressive and efficient: they can encode distributions with hundred millions of parameters and be effectively learned by gradient ascent [Peharz et al., 2020b]. The structure of their computational graph can be either specified manually [Poon and Domingos, 2011b; Peharz et al., 2020b,a] or acquired automatically from data [Vergari et al., 2015; Rahman et al., 2014b; Dang et al., 2022], e.g., by first learning a latent tree model and then compiling the latter into a circuit [Liu and Van den Broeck, 2021b]. These circuits are competitive with intractable models such as variational autoencoders and normalizing flows scores on several benchmarks [Liu et al., 2022].

Furthermore, one advantage PCs provide is that at each round of the greedy search we do  $O(n)$  independent calls to marginals, so we can batch the queries together and compute in parallel using GPUs. The size of the circuit is constant and controllable by us, hence the main bottleneck is the number of outlier  $k$ . However, this limitation might not be a big issue in practice since we assumed  $k$  is relatively small compared to  $n$ , and the computation cost only grows linearly w.r.t.  $k$ . For example, for our MNIST experiments, and  $k = 60$ , finding the outlier subset takes about 0.5 - 0.75 seconds per image (depending on the dataset) which seems reasonable. One potential avenue for getting an algorithm with runtime independent of  $k$  is exploring continuous relaxation of our submodular optimization tasks.

## 6.5 Continuous Relaxation of Search for Outlier Features

As we saw in previous sections, we were able to speed up the discrete search using submodular (supermodular) properties. However, we still need to evaluate  $O(N)$  new candidates for each

iteration and we have  $k$  iterations, so overall we have to do  $O(N \cdot k)$  evaluations during the discrete search. This motivates us to think about relaxing the discrete subset selection search into a continuous problem. In this section, we show how we can relax the discrete search into a continuous search problem using multi-linear extension.

**Definition 20** (Multi-linear Extension). *Given a set function  $f : 2^{|N|} \rightarrow \mathbb{R}$ , its multi-linear extension  $F : [0, 1]^n \rightarrow \mathbb{R}$  is defined as:*

$$F(\alpha) = \sum_{S \subseteq N} f(S) \prod_{i \in S} \alpha_i \prod_{i \notin S} (1 - \alpha_i) \quad (6.5.1)$$

**Probabilistic Interpretation of Multi-Linear Extension** Interestingly, the multi-linear extension extension can be also thought of as an expectation of  $f(a)$  where distribution of  $A = a$  is determined by having  $i \in S$  with probability  $\alpha_i$ , similarly for  $i \notin S$  we have probability  $(1 - \alpha_i)$ , more formally we have:

$$F(\alpha) = \sum_{S \subseteq N} f(S) \cdot q_\alpha(S) = \mathbb{E}_{S \sim q_\alpha} [ f(S) ] \quad (6.5.2)$$

where  $q_\alpha(S) = \prod_{i \in S} \alpha_i \prod_{i \notin S} (1 - \alpha_i)$ . Values for  $\alpha_i$  are also in range  $[0, 1]$ .

**Multi-Linear Extension of Circuits** There are a few challenges with multi-linear extension, firstly computing it is not easy in general for arbitrary set functions  $f$ . However, the good news is that if the set functions  $f(S)$  is defined as marginals of probabilistic circuits, then computing the multi-linear extension becomes tractable with some minor modification in the computation of the circuit's input nodes. We call this new class of modified circuits as alpha-relaxed circuits.

**Definition 21** (Alpha Relaxed Circuits). *Given the original circuit  $p$ , and output of leaf node  $n$  denoted by  $p_n$ . Now, the alpha-relaxed probabilistic circuit  $\hat{p}(\cdot; \alpha)$  is defined as follows: The sum and product nodes are defined the same way as usual in a probabilistic circuit, but for input nodes*

we do the following instead:

$$\hat{p}_n(\mathbf{x}_i; \alpha) = \alpha_i \cdot g_n(\mathbf{x}_i) + (1 - \alpha_i) \cdot 1 \quad (6.5.3)$$

Where  $g_n(\mathbf{x}_i)$  is the univariate distribution in the input node  $n$ .

The motivation for this definition comes from the probabilistic interpretations of the multi-linear extension. Here  $\alpha_i$  denotes probability of variable  $X_i$  being observed, when  $X_i$  is observed we get  $p_n(x_i)$  as usual, otherwise we output 1 (because for input nodes marginal probability of not observing a feature is 1). So overall  $\hat{p}_n$  is defined by:

$$\hat{p}_n(\mathbf{x}; \alpha) = \begin{cases} \alpha_i \cdot g_n(\mathbf{x}_i) + (1 - \alpha_i) \cdot 1 & \text{if } n \text{ is a distribution unit} \\ \prod_{c \in \text{ch}(n)} \hat{p}_c(\mathbf{x}; \alpha) & \text{if } n \text{ is a product unit} \\ \sum_{c \in \text{ch}(n)} \theta_{n,c} \cdot \hat{p}_c(\mathbf{x}; \alpha) & \text{if } n \text{ is a sum unit} \end{cases}$$

Next, we observe that computing marginals of alpha-relaxed circuits gives us the multi-linear extension of the original circuit:

**Theorem 12.** *Given an assignment  $\mathbf{x}$ , and a probabilistic circuit  $p$ , the alpha-relaxed version of the circuit  $\hat{p}(\mathbf{x}; \alpha)$  is equivalent to multi-linear extension  $F(\alpha)$  for the function  $f(S) = p(\mathbf{x}_S)$ , where  $p(\mathbf{x}_S)$  is the marginal probability of only observing features  $S$  in the original circuit.*

**Theorem 13** (Convexity of Multi-linear Extension). *Given a submodular (supermodular) set function  $f(S)$ , its multi-linear extension  $F(\alpha)$  is convex (concave).*

Now, given Theorem 13, and the fact that marginal probability set functions are supermodular, we can conclude that optimizing  $\alpha$  becomes a convex/concave optimization task. With also observing Theorem 12, we can convert our original discrete search problem into a continuous convex optimization with constraints. Next, we define the new optimization task.

**Definition 22** (Continuous Optimization Task). *: Given a probabilistic circuit  $p$ , and its alpha-*

relaxed equivalent  $\hat{p}$ . We can optimize the following task to find the outlier features.

$$\begin{aligned} & \arg \max_{\alpha} \hat{p}(\mathbf{x}; \alpha) \\ & \text{s.t. } (\forall i, 0 \leq \alpha_i \leq 1) \text{ and } \left( \sum_i \alpha_i \geq N - k \right) \end{aligned}$$

We can use the gradients to optimize alphas, however we need few extra steps to do the optimization, since we have both box constraints ( $0 \leq \alpha_i \leq 1$ ), and sum constraints ( $\sum_i \alpha_i \geq N - k$ ). We used the projected gradient method with small variation to optimize  $\alpha_i$ . In summary, we first compute gradients for alphas, then take one gradient step, finally in the end will make sure sum of alphas does not exceed the budget by dividing all alphas by a constant so the sum becomes  $N - K$ .

Our initial goal was to choose a subset of features as outliers, however after optimizing  $\alpha$  does not give us a clear subset. One simple way to go from alphas to subset is to choose the features with lowest  $k$  alphas. Although not perfect, it works well in practice. We leave exploration of other methods to go back from continuous to discrete outcome for future work. One promising avenue could be combining this method with subset- $k$  gradient estimate methods such as SIMPLE [Ahmed et al., , 2023].

## 6.6 Experiments

In this section, we evaluate both discrete and continuous search strategies mentioned in previous sections. First, we evaluate the discrete search for outliers. We provide both quantitative and qualitative results showcasing the performance of the discrete search. Next, we evaluate the quality of the continuous relaxation. In summary, the continuous search can accelerate the search by orders of magnitude in high dimensional settings, and also perform better than discrete search in some common low dimensional benchmarks.



Figure 6.1: Effective outlier feature detection with inlier maximization with a fixed budget. Each row corresponds to (top to bottom): original images, corrupted images, outliers found with budget  $k = 60$  (denoted with red pixels), and images that have been cleaned by the generative model. Each column corresponds to a different corruption process (left to right): Square(3x3), Square(5x5), Square(7x7), Square(9x9), Smiley, Column(14), Row(14), Random(120), Random(200).

Table 6.1: Datasets

Name	# Features	# Train	# Test
MNIST [Lecun et al., 1998]	784	60000	10000
FASHION [Xiao et al., 2017a]	784	60000	10000
EMNIST Letters [Cohen et al., 2017]	784	124800	20800

## Discrete Search Experiments

In this section, we showcase a few experiments to answer the questions: i) Is the discrete search method decent at finding outlier features in different corruption scenarios? ii) Can finding and fixing a subset of outlier features help in downstream tasks such as classification?

**Setup** In all the experiments, we only utilize one CPU core and one GPU (NVIDIA A5000). We choose three datasets MNIST [Lecun et al., 1998], EMNIST Letters [Cohen et al., 2017], and FASHION [Xiao et al., 2017a] to test generality of our method. For each dataset we use the default train and test splits provided. Table 6.1 provides more info about each dataset. For fair comparison, we do not want to tell our method exactly how many outliers they are in each scenario, so we choose a fixed budget for  $k$ . Unless otherwise noted, we choose  $k = 60$  throughout our experiments.

**Preprocessing Steps** For each dataset we assume there was no corruptions in the training data. We learn a probabilistic circuit using hidden Chow-Liu Tree (HCLT) algorithm [Liu and Van den Broeck, 2021a]. Additionally, we train a classifier for each dataset using Convolutional Neural Network architecture [LeCun et al., 1998]. Training the circuits are a one time cost for each dataset taking about 5-10 minute each.

**Corruption Scenarios** We try a wide variety of corruption patterns to showcase generality of our methods. Square( $i$ ) refers to a square corruption patch of size  $i$  by  $i$  relatively close to middle of the image. Row( $i$ ) or Column( $j$ ) refers to corrupting  $i$ 'th row or  $j$ 'th column respectively. Random( $i$ ) refers to choosing random pixels to corrupt repeated for  $i$  times. Smiley refers to corruption

Table 6.2: Percentage and Avg Outliers found for Maximizing Marginals of Inliers ( $k = 60$ )

CORRUPTION	OUTLIERS	MNIST		FASHION		EMNIST LETTERS	
		%	FOUND	%	FOUND	%	FOUND
SQUARE(3X3)	9	97	8.7	70	6.3	99	8.9
SQUARE(5X5)	25	86	21.5	56	13.9	87	21.7
SQUARE(7X7)	49	69	33.7	51	24.9	76	37.2
SQUARE(9X9)	81	54	44.1	38	30.5	63	51.0
SMILEY	89	52	46.0	44	38.7	59	52.8
COLUMN(14)	28	88	24.7	62	17.3	93	25.9
ROW(14)	28	91	25.4	81	22.8	97	27.0
RANDOM(120)	110	55	59.0	52	57.0	54	59.9
RANDOM(200)	180	33	59.9	33	58.1	32	60.0

consisting of 2 squares for the eyes and one wide rectangle for the mouth. More details about the corruption scenarios will be provided in the appendix.

Tables 6.2 and 6.3 summarize the results for finding the outlier features in each scenario and optimization method. For each dataset, the first column (%) shows the percentage of corrupted found successfully, and the second column (“Found”) shows the average number of corrupted features found by the algorithm. As we see, given our fixed budget of  $k = 60$ , both methods successfully find high percentage of outliers when there is less than 60 outliers. In general, the maximizing inliers method does better at finding the outliers specially in bigger patch shapes, almost all of the budget is correctly used on finding corrupted features. Figure 6.1 provides few a image examples of our process by showing the clean images randomly samples from the test data, their corruption (one column for each corruption type), outliers found by our method, and the corresponding fixed image.

**Fixing Outlier Features** In certain downstream tasks such as classification, in addition to finding the outlier features, we might need to also fix the outlier features. The first step is to find the outliers using either methods from Section 6.2. Since the main focus of the paper is finding the outlier subset, we use a fairly simple multiple imputation method to fix the the outlier features. We reuse the probabilistic circuit we learned from the data to generate the multiple imputations, since PCs



Table 6.3: Percentage and Avg Outliers found for Minimizing Marginals of Outliers ( $k = 60$ )

CORRUPTION	OUTLIERS	MNIST		FASHION		EMNIST LETTERS	
		%	FOUND	%	FOUND	%	FOUND
SQUARE(3X3)	9	83	7.5	73	6.5	81	7.3
SQUARE(5X5)	25	80	20.0	57	14.2	77	19.4
SQUARE(7X7)	49	71	35.0	43	20.9	67	32.8
SQUARE(9X9)	81	53	42.8	35	28.0	56	45.3
SMILEY	89	46	40.9	33	29.6	48	42.5
COLUMN(14)	28	72	20.2	41	11.5	87	24.4
ROW(14)	28	70	19.6	48	13.5	74	20.7
RANDOM(120)	110	40	43.6	31	35.1	41	44.8
RANDOM(200)	180	29	52.3	22	39.2	27	48.5

also allow for fast conditional sampling of the form  $p(\mathbf{X}_{out} | \mathbf{x}_{in})$ . We do 100 conditional samples for each corrupted image and replace the outlier features with the sampled features from the circuit, then we input all the 100 samples into our classifier and use the average logits to perform the classification. We record the accuracy before and after fixing the images, results are summarized in Tables 6.4 and 6.5.

Except a few cases, our method of finding and fixing outliers gives a decent boost to classification accuracy of the classifier on the corrupted test data. In some cases, we see performance boosts as high as 30% better accuracy. However, our methods do not always increase the classification accuracy, noticeably for bigger 9x9 square patches of MNIST dataset we even see a noticeable decrease in accuracy after replacing the outlier features. One reason could be due to having more corruptions (81) than our budget. Upon closer inspection of the fixed images for that scenario we noticed that the outlier detection does a decent job at finding the outlier features, however during imputation the outlier feature we did not find mess up the imputations. One might solve this issue by using more complex methods for fixing the outliers such as in-painting methods.

Table 6.4: Accuracy Gains for Maximizing Marginals of Inliers ( $k = 60$ )

CORRUPTION	MNIST			FASHION			EMNIST LETTERS		
	CORRUPT	FIXED	DIFF	CORRUPT	FIXED	DIFF	CORRUPT	FIXED	DIFF
SQUARE(3X3)	98.6	97.2	-1.4	83.5	86.1	+2.6	89.1	90.5	+1.3
SQUARE(5X5)	97.3	96.8	-0.5	79.1	82.1	+3.0	84.5	90.1	+5.6
SQUARE(7X7)	90.0	90.3	+0.3	79.8	79.4	-0.4	79.5	85.5	+6.0
SQUARE(9X9)	72.3	65.8	-6.6	76.0	75.9	-0.1	70.1	77.5	+7.4
SMILEY	84.2	83.9	-0.3	74.1	76.8	+2.7	60.4	74.7	+14.3
COLUMN(14)	97.7	97.6	-0.1	83.3	87.7	+4.4	86.2	90.3	+4.1
ROW(14)	96.2	98.1	+1.9	86.5	87.5	+1.0	83.4	91.0	+7.6
RANDOM(120)	87.7	97.3	+9.6	76.6	87.5	+10.9	54.7	86.6	+31.8
RANDOM(200)	72.6	81.9	+9.3	50.2	80.3	+30.1	25.9	46.2	+20.4

Table 6.5: Accuracy Gains for Minimizing Marginals of Outliers ( $k = 60$ )

CORRUPTION	MNIST			FASHION			EMNIST LETTERS		
	CORRUPT	FIXED	DIFF	CORRUPT	FIXED	DIFF	CORRUPT	FIXED	DIFF
SQUARE(3X3)	96.7	97.7	+1.0	86.5	88.3	+1.8	88.5	90.8	+2.3
SQUARE(5X5)	95.9	95.5	-0.4	83.0	84.5	+1.5	86.0	88.0	+2.0
SQUARE(7X7)	86.0	87.3	+1.2	75.4	76.9	+1.5	82.0	83.9	+1.9
SQUARE(9X9)	72.4	69.2	-3.2	74.1	74.3	+0.2	68.6	67.6	-1.0
SMILEY	79.4	80.4	+1.1	73.6	73.9	+0.4	57.9	59.2	+1.3
COLUMN(14)	97.2	97.4	+0.2	84.3	86.3	+2.0	87.5	91.3	+3.8
ROW(14)	93.7	97.7	+4.0	87.6	87.4	-0.2	79.8	89.5	+9.7
RANDOM(120)	94.8	96.8	+2.0	69.6	78.4	+8.8	54.7	70.1	+15.4
RANDOM(200)	85.5	87.8	+2.3	47.9	52.7	+4.8	20.3	25.1	+4.8

Table 6.6: Outlier Aspect Mining Baselines. Each dataset has 3 labeled ground truth. Each row compares the F1 score across different methods for the corresponding outlier ground truth.

DATASET	OURS		BASELINES		
	DISCRETE	ALPHAS	SPN-FW	SPN-BW	ATON
ARRHYTHMIA	0.652	0.628	<b>0.742</b>	0.726	0.676
	<b>0.692</b>	0.635	0.635	0.577	0.596
	<b>0.757</b>	0.723	0.695	0.751	0.557
ISONPHERE	0.632	0.630	<b>0.644</b>	0.488	0.622
	0.591	0.629	0.590	0.452	<b>0.671</b>
	0.536	0.570	<b>0.658</b>	0.564	0.618
LETTERS	0.700	<b>0.703</b>	0.701	0.519	0.665
	0.839	<b>0.844</b>	0.641	0.388	0.664
	0.745	0.747	<b>0.778</b>	0.752	0.545
OPTDIGITS	0.789	<b>0.791</b>	0.754	0.450	0.671
	<b>0.885</b>	<b>0.885</b>	0.725	0.472	0.672
	0.741	0.758	<b>0.887</b>	0.871	0.557
PIMA	0.686	<b>0.690</b>	0.589	0.538	0.673
	<b>0.653</b>	0.651	0.632	0.515	0.65
	0.638	0.637	<b>0.747</b>	0.656	0.531
SATIMAGE	<b>0.645</b>	0.641	0.604	0.612	0.585
	0.330	0.550	0.661	0.590	<b>0.664</b>
	0.510	0.546	0.746	<b>0.823</b>	0.541
WBC	0.600	0.650	<b>0.718</b>	0.630	0.604
	0.544	0.565	0.552	0.447	<b>0.601</b>
	0.615	0.654	<b>0.679</b>	0.659	0.579
WINERED	0.667	<b>0.683</b>	0.436	0.366	0.661
	0.579	0.611	0.432	0.367	<b>0.652</b>
	0.585	<b>0.616</b>	0.491	0.407	0.481
WINEWHITE	0.673	<b>0.721</b>	0.526	0.454	0.619
	0.572	0.561	0.469	0.428	<b>0.605</b>
	0.695	<b>0.712</b>	0.569	0.529	0.479

Table 6.7: Summary of Table 6.6 Results. Each entry in the table denotes how often does the method get better F1 results than others.

METHOD	OURS		BASELINES		
	DISCRETE	ALPHAS	SPN-FW	SPN-BW	ATON
ALPHAS	70%	-	56%	70%	74%

## Continuous Search Experiments

**Setup:** Firstly, we repeated the experiments on the datasets from previous section to measure runtime boost and compare performance. Next, we evaluate both discrete and continuous versions of the search on a benchmark introduced in Liu et al. [2018] with 9 real world datasets with ground truth labels indicating which features are outliers.

**Real world Datasets:** We compare our results with a few popular baselines such as SPN-bw, SPN-fw [Lüdtke et al., 2023], ATON [Xu et al., 2021], and COIN [Liu et al., 2018]. In Table 6.6, we include the results for each dataset and outlier explanation method. Each dataset has three sets of ground truth labels. For each scenario we report the F1 score of the outlier explanation method on the corresponding labeling method. In Table 6.7, we summarize the results by measuring how often continuous relaxation (Alpha’s method) does better than other methods. Although there is no method that is always the best, the Alphas method beats others more than 50%-70% of the time.

**Speed Gains on MNIST:** For the Alphas method, the gain in speed would depend on factors size as number of features, and number of outliers. In practice, it could lead to orders of magnitude speed up. For example, for the MNIST benchmarks, the alphas method was on average more than 80 times faster, leading to similar qualitative results in identifying outlier pixels. In terms of F1 score, also the Alphas method was at most 4% worst, while in some cases it was better.

## 6.7 Related Work

The idea of using generative models for fixing adversarial perturbations of images to improve accuracy of classifiers has been widely explored, for example PixelDefend [Song et al., 2018] uses PixelCNN models to fix outlier images by bringing closer to the original distribution, more recently DiffPure [Nie et al., 2022] uses diffusion models to purify adversarial perturbations. In many of these approaches, the assumption is that the perturbations do not change pixel values by too much, however they generally allow all features to be changed slightly. In contrast, we allow arbitrary changes to pixel values but limit the number of corrupted pixels.

Taking advantage of Submodularity of information theoretic measures such as entropy and mutual information in machine learning has been explored before [Iyer et al., 2021]. To the best of our knowledge, we are the first to find applications for submodularity of marginal probability distributions in machine learning scenarios.

Both defences and attacks on fixed shape patches (such as squares) with arbitrary perturbations of pixels has been widely explored [Levine and Feizi, 2020; Chiang\* et al., 2020; Brown et al., 2017].

## Conclusion

In this paper, we provided two novel algorithms for the challenging subset selection problem of finding subset of outlier features. We take advantage of supermodularity and monotonicity of marginal probability distributions to formulate the problem as a monotonic submodular optimization task which are generally easier to optimize. We empirically show that the proposed methods are promising at finding and explaining outliers, and can help increase accuracy of classifiers on corrupted data. We hope this paper will motivate more explorations on taking advantage of marginal likelihoods in finding outlier features.

# Appendix A

## Reasoning about Logistic Regression with Missing Features

### A.1 Proofs

#### A.1.1 Proof of Theorem 1

The proof is by reduction from computing the same-decision probability, whose decision problem **D-SDP** was shown to be NP-hard. Chen et al. [2013]

Given a naive Bayes distribution  $P(\cdot)$  over variables  $C$  and  $\mathbf{X}$ , a threshold  $T$ , and a probability  $p$ , **D-SDP** asks: is the same-decision probability  $\sum_{\mathbf{x}} \mathbb{I}(P(c|\mathbf{x}) > T)P(\mathbf{x})$  greater than  $p$ ? Here,  $\mathbb{I}(\cdot)$  denotes an indicator function which returns 1 if the enclosed expression is true, and 0 otherwise.

Using Lemma 2 we can efficiently translate a naive Bayes model  $P$  to a logistic regression with a weight function  $w(\cdot)$  such that

$$P(c|\mathbf{x}) = \frac{1}{1 + e^{-w(\mathbf{x})}}.$$

Note that  $P(c|\mathbf{x}) > T$  iff  $w(\mathbf{x}) > -\log(\frac{1}{T} - 1)$ . Then we construct another logistic regression with weight function

$$w'(\mathbf{x}) = n \cdot \left( w(\mathbf{x}) + \log\left(\frac{1}{T} - 1\right) \right),$$

for some positive constant  $n$ . As  $w$  is a linear model,  $w'$  is also linear, and  $w'(\mathbf{x}) > 0$  iff  $P(c|\mathbf{x}) > T$ . As  $n$  grows,  $w'(\cdot)$  approaches  $\infty$  and  $-\infty$  for positive and negative examples, respectively. Hence, this logistic regression model outputs 1 if  $P(c|\mathbf{x}) > T$  and 0 otherwise, effectively being an indicator function. Therefore, the expectation of such classifier over  $P(\mathbf{X})$  is equal to the same-decision probability of  $\mathbf{X}$ . □

### A.1.2 Proof of Lemma 2

We want to prove there is a unique  $\mathcal{F}$  such that  $\mathcal{F}(\mathbf{x}) = P(c | \mathbf{x})$  for all  $\mathbf{x}$  given naive Bayes distribution  $P$ . Using Bayes' rule and algebraic manipulation, we get:

$$\begin{aligned} P(c | \mathbf{x}) &= \frac{P(\mathbf{x} | c) P(c)}{P(\mathbf{x} | c) P(c) + P(\mathbf{x} | \bar{c}) P(\bar{c})} \\ &= \frac{1}{1 + \frac{P(\mathbf{x} | \bar{c}) P(\bar{c})}{P(\mathbf{x} | c) P(c)}} = \frac{1}{1 + \exp \left[ -\log \frac{P(\mathbf{x} | c) P(c)}{P(\mathbf{x} | \bar{c}) P(\bar{c})} \right]} \end{aligned}$$

For any input  $\mathbf{x}$ , we want above quantity to be equal to  $\mathcal{F}(\mathbf{x}) = 1/(1 + \exp[-\sum_i w_i x_i])$ . In other words, we need:

$$\log \frac{P(\mathbf{x} | c) P(c)}{P(\mathbf{x} | \bar{c}) P(\bar{c})} = \sum_i w_i x_i$$

Using naive Bayes assumption, we arrive at:

$$\sum_{i=0}^n w_i x_i = \log \frac{P(c)}{P(\bar{c})} + \sum_{i=1}^n \log \frac{P(x_i | c)}{P(x_i | \bar{c})} \quad (\text{A.1.1})$$

Now we want the RHS of Equation A.1.1 to be a linear function of  $x_i$ 's, so we do the following substitution for  $i > 0$  assuming binary features:

$$\begin{aligned} \log \frac{P(x_i | c)}{P(x_i | \bar{c})} &= (x_i) \cdot \log \frac{P(x_i = 1 | c)}{P(x_i = 1 | \bar{c})} \\ &\quad + (1 - x_i) \cdot \log \frac{P(x_i = 0 | c)}{P(x_i = 0 | \bar{c})} \end{aligned} \quad (\text{A.1.2})$$

By combining Equations A.1.1 and A.1.2 we get the weights in Lemma 2 by simple algebraic manipulations. To solve for the bias term  $w_0$  we plug in  $x_i = 0$  for all  $i > 0$ . To compute  $w_i$  for a non-zero  $i$  we take the coefficient of  $x_i$  in Equation A.1.2.  $\square$

### A.1.3 Proof of Lemma 3

Through the same algebraic manipulation as before, we get the same equations as in Lemma 2 with the only difference being that we are now solving the parameters of a naive Bayes model rather than weights of the logistic regression model. Intuitively, because the NB model has  $2n + 1$  free parameters but the LR model only has  $n + 1$  parameters, we expect some degree of freedom. To get rid of the freedom and get a unique solution, we fix the values for  $n$  parameters as follows:

$$P(x_i = 1 | c) = \theta_i \tag{A.1.3}$$

Without loss of generality we have fixed the parameter values for positive features. One can equally set the values in other ways as long as one parameter value per feature is fixed.

Now there is a unique naive Bayes model that matches the logistic regression classifier and also agrees with Equation A.1.3. That is, the remaining  $n + 1$  parameter values are given by the LR parameters, resulting in the following parameters for such naive Bayes model:

$$\begin{aligned} P(x_i | c) &= \theta_i, & P(\bar{x}_i | c) &= 1 - P(x_i | c), \\ P(x_i | \bar{c}) &= \frac{1}{1 + e^{w_i \frac{1-\theta_i}{\theta_i}}}, & P(\bar{x}_i | \bar{c}) &= 1 - P(x_i | \bar{c}), \\ P(c) &= \text{sigmoid} \left( w_0 - \sum_{i=1}^n \log \frac{P(\bar{x}_i | c)}{P(\bar{x}_i | \bar{c})} \right). \end{aligned}$$

□

## A.2 Beyond Binary Classification: Multiclass

In the paper, we studied logistic regression and conformant naive Bayes models assuming binary classification. We now show that our method can easily be extended to multiclass. We first modify



our notation of logistic regression and naive Bayes models to allow for an arbitrary number of classes.

**Definition 23.** (*Multiclass Classifiers*) Suppose we have a classifier with  $K$  classes, each denoted by  $c_k$  ( $k \in [0, K - 1]$ ). Then  $\mathcal{F}_k$  denotes the conditional probability for class  $c_k$  in logistic regression, and  $P$  a naive Bayes distribution defined as:

$$\mathcal{F}_k(\mathbf{x}) = \frac{e^{W_k \cdot \mathbf{x}}}{\sum_j e^{W_j \cdot \mathbf{x}}}$$

$$P(c_k | \mathbf{x}) = \frac{P(c_k) \prod_{i=1}^n P(x_i | c_k)}{\sum_j P(c_j) \prod_{i=1}^n P(x_i | c_j)}$$

We say  $P$  conforms with  $\mathcal{F}$  if their predictions agree for all classes:  $P(c_k | \mathbf{x}) = \mathcal{F}_k(\mathbf{x})$  for all  $k$  and  $\mathbf{x}$ .

Next, we describe naive conformant learning for multiclass. Instead of directly matching the predictions of  $P$  and  $\mathcal{F}$  for all classes, we match their ratios in order to simplify equations going forward. Moreover, to get the same classifiers it suffices to divide by the probability of only one class, so without loss of generality we set the following be true.

$$\frac{\mathcal{F}_k(\mathbf{x})}{\mathcal{F}_0(\mathbf{x})} = \frac{P(c_k | \mathbf{x})}{P(c_0 | \mathbf{x})}, \quad \forall k \in [1, K - 1]$$

Using Definition 23, this leads to

$$\frac{e^{-W_k \cdot \mathbf{x}}}{e^{-W_0 \cdot \mathbf{x}}} = \frac{P(c_k) \prod_{i=1}^n P(x_i | c_k)}{P(c_0) \prod_{i=1}^n P(x_i | c_0)}, \quad \forall k \in [1, K - 1].$$

The parameters of a multiclass naive Bayes are:  $\theta_{c_k} = P(c_k)$ ,  $\theta_{x_i | c_k} = P(x_i = 1 | c_k)$ , and  $\theta_{\bar{x}_i | c_k} = P(x_i = 0 | c_k)$ . Then, we get the following constraints for NaCL through similar algebraic

manipulations as in the binary case:

$$\sum_k \theta_{c_k} = 1 \tag{A.2.1}$$

$$\theta_{x_i|c_k} + \theta_{\bar{x}_i|c_k} = 1, \quad \forall i, k > 0 \tag{A.2.2}$$

$$e^{w_{k,i}-w_{0,i}} \theta_{x_i|c_k}^{-1} \theta_{\bar{x}_i|c_k} \theta_{x_i|c_0} \theta_{\bar{x}_i|c_0}^{-1} = 1, \quad \forall i, k > 0$$

$$e^{w_{k,0}-w_{0,0}} \theta_{c_k}^{-1} \theta_{c_0} \prod_{i=1}^n \theta_{\bar{x}_i|c_k}^{-1} \theta_{\bar{x}_i|c_0} = 1, \quad \forall k > 0$$

Again, we relax Equations A.2.1 and A.2.2 to inequalities to obtain valid geometric program constraints. The rest of the method stays the same: we maximize the marginal likelihood with above constraints by minimizing the inverse of the joint likelihood on a completed dataset, as described in Section 2.4.

## Appendix B

### On Tractable Computation of Expected Predictions

#### B.1 Proofs

##### B.1.1 Proofs of Propositions 3.1 and 3.3

We will first prove Proposition 3.3, from which Proposition 3.1 directly follows. For a PC OR node  $n$  and RC OR node  $m$ ,

$$\begin{aligned}
 M_k(g_m, p_n) &= \mathbb{E}_{\mathbf{x} \sim p_n(\mathbf{x})} [g_m^k(\mathbf{x})] \\
 &= \mathbb{E}_{\mathbf{x} \sim p_n(\mathbf{x})} \left[ \left( \sum_{j \in \text{ch}(m)} \mathbf{1}_j(\mathbf{x}) (g_j(\mathbf{x}) + \phi_j) \right)^k \right] \\
 &= \mathbb{E}_{\mathbf{x} \sim p_n(\mathbf{x})} \sum_{j \in \text{ch}(m)} \left[ (\mathbf{1}_j(\mathbf{x}) (g_j(\mathbf{x}) + \phi_j))^k \right] \tag{B.1.1} \\
 &= \mathbb{E}_{\mathbf{x} \sim p_n(\mathbf{x})} \sum_{j \in \text{ch}(m)} \sum_{l=0}^k \binom{k}{l} g_j^l(\mathbf{x}) \phi_j^{k-l} \mathbf{1}_j(\mathbf{x}) \\
 &= \sum_{\mathbf{x} p_n(\mathbf{x})} \sum_{j \in \text{ch}(m)} \sum_{l=0}^k \binom{k}{l} g_j^l(\mathbf{x}) \phi_j^{k-l} \mathbf{1}_j(\mathbf{x}) \\
 &= \sum_{\mathbf{x} \sum_{i \in \text{ch}(n)} \theta_i p_i(\mathbf{x})} \sum_{j \in \text{ch}(m)} \sum_{l=0}^k \binom{k}{l} g_j^l(\mathbf{x}) \phi_j^{k-l} \mathbf{1}_j(\mathbf{x}) \\
 &= \sum_{i \in \text{ch}(n)} \theta_i \sum_{j \in \text{ch}(m)} \sum_{l=0}^k \binom{k}{l} \phi_j^{k-l} \sum_{\mathbf{x} p_i(\mathbf{x})} g_j^l(\mathbf{x}) \mathbf{1}_j(\mathbf{x})
 \end{aligned}$$

$$\begin{aligned}
&= \sum_{i \in \text{ch}(n)} \theta_i \sum_{j \in \text{ch}(m)} \sum_{l=0}^k \binom{k}{l} \phi_j^{k-l} \mathbb{E}_{\mathbf{x} \sim p_i(\mathbf{x})} [\mathbb{1}_j(\mathbf{x}) g_j^l(\mathbf{x})] \\
&= \sum_{i \in \text{ch}(n)} \theta_i \sum_{j \in \text{ch}(m)} \sum_{l=0}^k \binom{k}{l} \phi_j^{k-l} M_l(\mathbb{1}_j \cdot g_j, p_i). \tag{B.1.2}
\end{aligned}$$

Equation B.1.1 follows from determinism of RCs as at most one  $j$  will have a non-zero  $\mathbb{1}_j(\mathbf{x})$ . In Equation B.1.2, note that we denote, with slight abuse of notation,  $M_0(\mathbb{1}_j \cdot g_j, p_i) = \mathbb{E}_{\mathbf{x} \sim p_i(\mathbf{x})} [\mathbb{1}_j(\mathbf{x})] = M_1(\mathbb{1}_j, p_i)$ . This concludes the proof of Proposition 3.3.

We obtain Proposition 3.1 by applying above result with  $k = 1$ :

$$\begin{aligned}
M_1(g_m, p_n) &= \sum_{i \in \text{ch}(n)} \theta_i \sum_{j \in \text{ch}(m)} \sum_{l=0}^1 \binom{1}{l} \phi_j^{1-l} M_l(\mathbb{1}_j \cdot g_j, p_i) \\
&= \sum_{i \in \text{ch}(n)} \theta_i \sum_{j \in \text{ch}(m)} (\phi_j M_0(\mathbb{1}_j \cdot g_j, p_i) + M_1(\mathbb{1}_j \cdot g_j, p_i)) \\
&= \sum_{i \in \text{ch}(n)} \theta_i \sum_{j \in \text{ch}(m)} (\phi_j M_1(\mathbb{1}_j, p_i) + M_1(\mathbb{1}_j \cdot g_j, p_i)).
\end{aligned}$$

### B.1.2 Proofs of Proposition 3.2 and 3.4

Again, we will first prove Proposition 3.4. For a PC AND node  $n$  and RC AND node  $m$ ,

$$\begin{aligned}
M_k(\mathbb{1}_m g_m, p_n) &= \mathbb{E}_{\mathbf{x} \sim p_n(\mathbf{x})} [\mathbb{1}_m(\mathbf{x}) g_m^k(\mathbf{x})] \\
&= \mathbb{E}_{\mathbf{x} \sim p_n(\mathbf{x})} [\mathbb{1}_m(\mathbf{x}) (g_{m_{\mathcal{L}}}(\mathbf{x}^{\mathcal{L}}) + g_{m_{\mathcal{R}}}(\mathbf{x}^{\mathcal{R}}))^k] \\
&= \sum_{\mathbf{x}^{\mathcal{L}}, \mathbf{x}^{\mathcal{R}}} p_{n_{\mathcal{L}}}(\mathbf{x}^{\mathcal{L}}) p_{n_{\mathcal{R}}}(\mathbf{x}^{\mathcal{R}}) \mathbb{1}_m(\mathbf{x}) (g_{m_{\mathcal{L}}}(\mathbf{x}^{\mathcal{L}}) + g_{m_{\mathcal{R}}}(\mathbf{x}^{\mathcal{R}}))^k \\
&= \sum_{\mathbf{x}^{\mathcal{L}}, \mathbf{x}^{\mathcal{R}}} p_{n_{\mathcal{L}}}(\mathbf{x}^{\mathcal{L}}) p_{n_{\mathcal{R}}}(\mathbf{x}^{\mathcal{R}}) \mathbb{1}_{m_{\mathcal{L}}}(\mathbf{x}^{\mathcal{L}}) \mathbb{1}_{m_{\mathcal{R}}}(\mathbf{x}^{\mathcal{R}}) \sum_{l=0}^k \binom{k}{l} g_{m_{\mathcal{L}}}^l(\mathbf{x}^{\mathcal{L}}) g_{m_{\mathcal{R}}}^{k-l}(\mathbf{x}^{\mathcal{R}}) \tag{B.1.3} \\
&= \sum_{l=0}^k \binom{k}{l} \left( \sum_{\mathbf{x}^{\mathcal{L}}} p_{n_{\mathcal{L}}}(\mathbf{x}^{\mathcal{L}}) \mathbb{1}_{m_{\mathcal{L}}}(\mathbf{x}^{\mathcal{L}}) g_{m_{\mathcal{L}}}^l(\mathbf{x}^{\mathcal{L}}) \right) \left( \sum_{\mathbf{x}^{\mathcal{R}}} p_{n_{\mathcal{R}}}(\mathbf{x}^{\mathcal{R}}) \mathbb{1}_{m_{\mathcal{R}}}(\mathbf{x}^{\mathcal{R}}) g_{m_{\mathcal{R}}}^{k-l}(\mathbf{x}^{\mathcal{R}}) \right)
\end{aligned}$$

$$\begin{aligned}
&= \sum_{l=0}^k \binom{k}{l} \mathbb{E}_{\mathbf{x}^{\mathcal{L}} \sim p_{n_{\mathcal{L}}}(\mathbf{x}^{\mathcal{L}})} [\mathbb{1}_{m_{\mathcal{L}}}(\mathbf{x}^{\mathcal{L}}) g_{m_{\mathcal{L}}}^l(\mathbf{x}^{\mathcal{L}})] \mathbb{E}_{\mathbf{x}^{\mathcal{R}} \sim p_{n_{\mathcal{R}}}(\mathbf{x}^{\mathcal{R}})} [\mathbb{1}_{m_{\mathcal{R}}}(\mathbf{x}^{\mathcal{R}}) g_{m_{\mathcal{R}}}^{k-l}(\mathbf{x}^{\mathcal{R}})] \\
&= \sum_{l=0}^k \binom{k}{l} M_l(\mathbb{1}_{m_{\mathcal{L}}} \cdot g_{m_{\mathcal{L}}}, p_{n_{\mathcal{L}}}) M_{k-l}(\mathbb{1}_{m_{\mathcal{R}}} \cdot g_{m_{\mathcal{R}}}, p_{n_{\mathcal{R}}}).
\end{aligned}$$

Equation B.1.3 follows from decomposability:  $\mathbb{1}_m(\mathbf{x}) = \mathbb{1}\{\mathbf{x} \models [m]\} = \mathbb{1}\{\mathbf{x} \models [m_{\mathcal{L}} \wedge m_{\mathcal{R}}]\} = \mathbb{1}\{\mathbf{x}^{\mathcal{L}} \models [m_{\mathcal{L}}]\} \mathbb{1}\{\mathbf{x}^{\mathcal{R}} \models [m_{\mathcal{R}}]\} = \mathbb{1}_{m_{\mathcal{L}}}(\mathbf{x}^{\mathcal{L}}) \mathbb{1}_{m_{\mathcal{R}}}(\mathbf{x}^{\mathcal{R}})$ . This concludes the proof of Proposition 3.4.

We obtain Proposition 3.2 by combining above result at  $k = 1$  with Equation 3.4.1:

$$\begin{aligned}
&M_1(\mathbb{1}_m \cdot g_m, p_n) \\
&= \sum_{l=0}^1 \binom{1}{l} M_l(\mathbb{1}_{m_{\mathcal{L}}} \cdot g_{m_{\mathcal{L}}}, p_{n_{\mathcal{L}}}) M_{1-l}(\mathbb{1}_{m_{\mathcal{R}}} \cdot g_{m_{\mathcal{R}}}, p_{n_{\mathcal{R}}}) \\
&= M_0(\mathbb{1}_{m_{\mathcal{L}}} \cdot g_{m_{\mathcal{L}}}, p_{n_{\mathcal{L}}}) M_1(\mathbb{1}_{m_{\mathcal{R}}} \cdot g_{m_{\mathcal{R}}}, p_{n_{\mathcal{R}}}) + M_1(\mathbb{1}_{m_{\mathcal{L}}} \cdot g_{m_{\mathcal{L}}}, p_{n_{\mathcal{L}}}) M_0(\mathbb{1}_{m_{\mathcal{R}}} \cdot g_{m_{\mathcal{R}}}, p_{n_{\mathcal{R}}}) \\
&= M_1(\mathbb{1}_{m_{\mathcal{L}}}, p_{n_{\mathcal{L}}}) M_1(g_{m_{\mathcal{R}}}, p_{n_{\mathcal{R}}}) + M_1(\mathbb{1}_{m_{\mathcal{R}}}, p_{n_{\mathcal{R}}}) M_1(g_{m_{\mathcal{L}}}, p_{n_{\mathcal{L}}}).
\end{aligned}$$

### B.1.3 Proof of Theorem 5

The proof is by reduction from the model counting problem (#SAT) which is known to be #P-hard.

Given a CNF formula  $\alpha$ , let us construct  $\beta$  and  $\gamma$  as follows. For every variable  $X_i$  appearing in clause  $\alpha_j$ , introduce an auxiliary variable  $X_{ij}$ . Then:

$$\begin{aligned}
\beta &\equiv \bigwedge_i (X_{i1} \Leftrightarrow \cdots \Leftrightarrow X_{ij} \Leftrightarrow \cdots \Leftrightarrow X_{im}), \\
\gamma &\equiv \bigwedge_j \bigvee_i l_{\alpha}(X_{ij}).
\end{aligned}$$

Here,  $l_{\alpha}(X_{ij})$  denotes the literal of  $X_i$  (i.e.,  $X_i$  or  $\neg X_i$ ) in clause  $\alpha_j$ . Thus,  $\gamma$  is the same CNF formula as  $\alpha$ , except that a variable in  $\alpha$  appears as several different copies in  $\gamma$ . The formula  $\beta$  ensures that the copied variables are all equivalent. Thus, the model count of  $\alpha$  must equal the model count of  $\beta \wedge \gamma$ .

Consider a right-linear vtree in which variables appear in the following order:  $X_{11}, X_{12}, \dots, X_{1j}, \dots, X_{ij}, \dots$ . The PC sub-circuit involving copies of variable  $X_i$  has exactly two model and size that is linear in the number of copies. There are as many such sub-circuits as there are variables in the original formula  $\alpha$ , each of which can be chained together directly to obtain  $\beta$ . The key insight in doing so is that sub-circuits corresponding to different variables  $X_i$  are independent of one another. Then, we can construct a PC circuit structure whose logical formula represents  $\beta$  in polytime. In a single top down pass, we can parameterize the PC  $p_n$  such that it represents a uniform distribution: each model is assigned a probability of  $1/2^n$ .

Next, consider a right-linear vtree with the variables appearing in the following order:

$$X_{11}, X_{21}, \dots, X_{n1}, \dots, X_{ij}, \dots$$

Then, we can construct a logical circuit that represents  $\gamma$  in polynomial time, as each variable appears exactly once in the formula. That is, each clause  $\alpha_j$  will have a PC sub-circuit with linear size (in the number of literals appearing in the clause), and the size of their conjunction  $\alpha$  will simply be the sum of the sizes of such sub-circuits. We can parameterize it as a regression circuit  $g_m$  by assigning 0 to all inputs to OR gates and adding a single OR gate on top of the root node with a weight 1. Then this regression circuit outputs 1 if and only if the input assignment satisfies  $\gamma$ .

Then the expectation of regression circuit  $g_m$  w.r.t. PC  $p_n$  (which does not share the same vtree) can be used to compute the model count of  $\alpha$  as follows:

$$\begin{aligned} M_1(g_m, p_n) &= \mathbb{E}_{\mathbf{x} \sim p_n(\mathbf{x})}[g_m(\mathbf{x})] = \sum_{\mathbf{x}} p_n(\mathbf{x}) g_m(\mathbf{x}) = \sum_{\mathbf{x}} \frac{1}{2^n} \mathbb{1}[\mathbf{x} \models \beta] \mathbb{1}[\mathbf{x} \models \gamma] \\ &= \frac{1}{2^n} \sum_{\mathbf{x}} \mathbb{1}[\mathbf{x} \models \beta \wedge \gamma] = \frac{1}{2^n} \text{MC}(\beta \wedge \gamma) = \frac{1}{2^n} \text{MC}(\alpha) \end{aligned}$$

Thus, #SAT can be reduced to the problem of computing expectations of a regression circuit w.r.t. a PC that does not share the same vtree. □

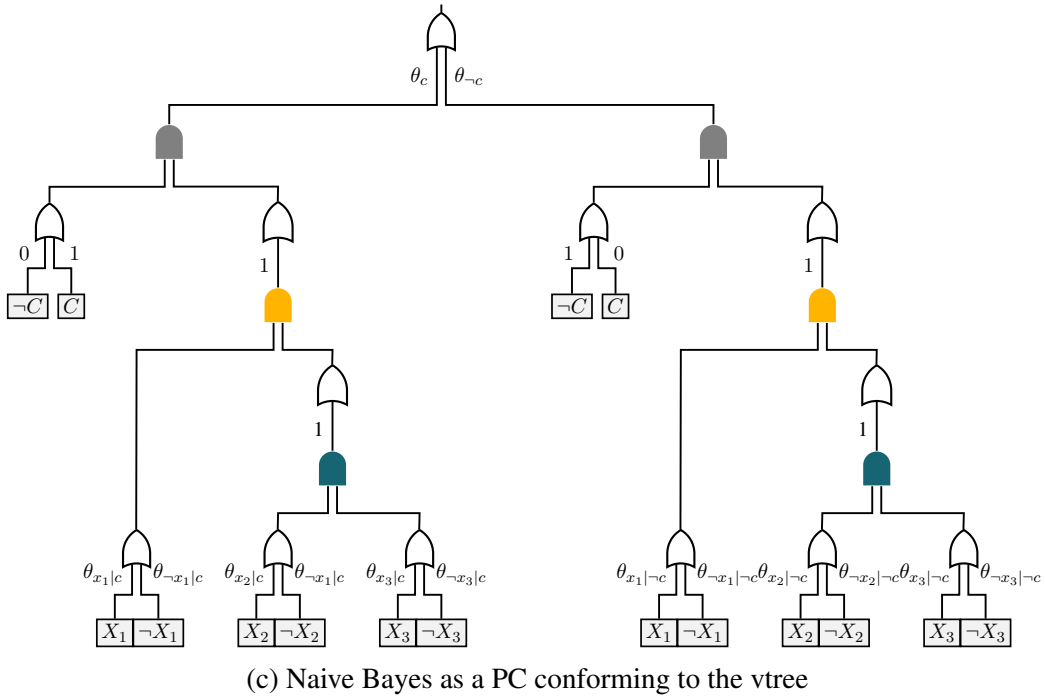
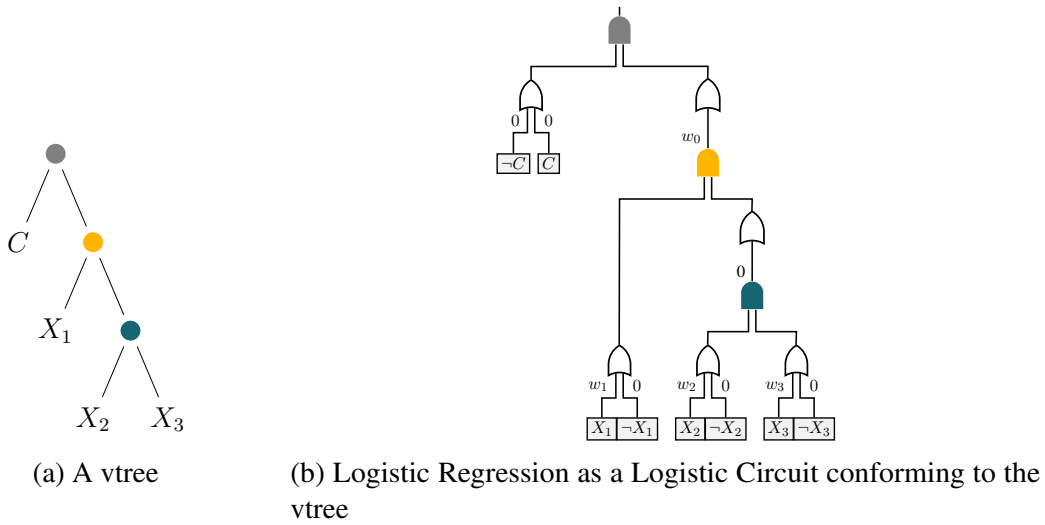


Figure B.1: A vtree (a) over  $\mathbf{X} = \{X_1, X_2, X_3\}$  and corresponding circuits that are respectively equivalent to a given Logistic Regression model with parameters  $w_0, w_1, w_2, w_3$ , and a Naive Bayes model with parameters  $\theta_c, \theta_{x_i|c}, \theta_{x_i|\sim c}$ .

### B.1.4 Proof of Theorem 6

The proof is by reduction from computing expectation of a logistic regression w.r.t. a naive Bayes distribution, which was shown to be NP-hard.

Given a naive Bayes distribution  $P(\mathbf{X}, C)$ , we can build a PC  $p_n$  that represents the same distribution in polynomial time by employing a right-linear vtree in which the class variable appears at the top, followed by the features. Because the feature distribution conditioned on the class variable is fully factorized, the PC sub-circuits corresponding to  $P(\mathbf{X}|C)$  and  $P(\mathbf{X}|\neg C)$  will each have size that is linear in the number of features.

Moreover, given a logistic regression model  $f(\mathbf{x}) = \sigma(w(\mathbf{x}))$ , we can build a corresponding logistic circuit  $\sigma \circ g_m$  in polytime using the same vtree as the PC described previously. Specifically, each non-leaf node  $v$  in the vtree corresponds to an AND gate, and for each its child we add an OR gate with parameter 0 (to keep the structure of alternating between AND and OR gates), recursively building the circuit. The leaf node for each variable  $X$  become an OR gate with 2 children  $X$  and  $\neg X$ , with parameters  $w_i$  and 0, respectively. The leaf nodes involving the class variable  $C$  will simply have weights 0. As shown in Liang and Van den Broeck [2019a], logistic circuits become equivalent to logistic regression on the feature embedding space, define by the structure of the circuit, as well as the “raw” features. With this parameterization, we ensure that the extra features introduced by the logistic circuit structure always have weight 0, so overall the circuit becomes equivalent to the original logistic regression. That is,  $w(\mathbf{x}) = g_m(C, \mathbf{x}) = g_m(\neg C, \mathbf{x})$  for all assignments  $\mathbf{x}$ .

Figure B.1 gives an example of the construction of the circuits using a given vtree, logistic regression, and a naive Bayes model. The logistic regression model is defined as  $f(\mathbf{x}) = \sum_i \mathbf{x}_i w_i$ , and for the naive Bayes model parameters are  $\theta_c = P(c)$ ,  $\theta_{x_i|c} = P(\mathbf{x}_i | c)$ , and  $\theta_{x_i|\neg c} = P(\mathbf{x}_i | \neg c)$ . Other values can be easily computed using the complement rule, for example  $\theta_{\neg x_i|c} = 1 - \theta_{x_i|c}$ . Finally, the naive Bayes distribution is now defined as:  $P(\mathbf{x}, C) = \theta_C \prod_i \theta_{x_i|C}$ .

The expectation of such logistic circuit  $\sigma \circ g_m$  w.r.t. PC  $p_n$  is equal to the expectation of original logistic regression  $f$  w.r.t. naive Bayes  $P$  as the following:

$$M_1(\sigma \circ g_m, p_n) = \mathbb{E}_{c\mathbf{x} \sim p_n(c\mathbf{x})}[\sigma(g_m(c\mathbf{x}))] = \mathbb{E}_{c\mathbf{x} \sim P(c\mathbf{x})}[f(\mathbf{x})] = \mathbb{E}_{\mathbf{x} \sim P(\mathbf{x})}[f(\mathbf{x})].$$



□

### B.1.5 Approximating expected prediction of classifiers

In this section, we provide more intuition on how we derived our approximation method for the case of classification. As mentioned in the main text, we define the following  $d$ -order approximation:

$$T_d(\gamma \circ g_m, p_n) \triangleq \sum_{k=0}^d \frac{\gamma^{(k)}(\alpha)}{k!} M_k(g_m - \alpha, p_n)$$

We can use  $T_d(\gamma \circ g_m, p_n)$  as an approximation to  $M_1(\gamma \circ g_m, p_n)$  because:

$$\begin{aligned} M_1(\gamma \circ g_m, p_n) &= \mathbb{E}_{\mathbf{x} \sim p_n(\mathbf{x})} [\gamma(g_m(\mathbf{x}))] = \mathbb{E}_{\mathbf{x} \sim p_n(\mathbf{x})} \sum_{i=0}^{\infty} \frac{\gamma^{(i)}(\alpha)}{i!} (g_m(\mathbf{x}) - \alpha)^i \\ &\approx \sum_{i=0}^d \frac{\gamma^{(i)}(\alpha)}{i!} \mathbb{E}_{\mathbf{x} \sim p_n(\mathbf{x})} (g_m(\mathbf{x}) - \alpha)^i = T_d(\gamma \circ g_m, p_n) \end{aligned}$$

For example, given a PC with root  $n$  and a logistic circuit with root  $m$  and sigmoid activation, the Taylor series around point  $\alpha = 0$  and  $d = 5$  gives us:

$$M_1(\gamma \circ g_m, p_n) \approx T_5(\gamma \circ g_m, p_n) = \frac{1}{2} + \frac{M_1(g_m, p_n)}{4} - \frac{M_3(g_m, p_n)}{48} + \frac{M_5(g_m, p_n)}{480}$$

In general, we would like to expand the Taylor series around a point that converges quickly. In our case, we employ  $\alpha \approx M_1(g_m, p_n)$ . All these Taylor expansion terms can be computed efficiently, as long as taking the derivatives of our non-linearity can be done efficiently at point  $\alpha$ .

## B.2 Datasets

We employed the following datasets for our empirical evaluation, taken from the UCI Machine Learning repository and other regression Khiari et al. [2018] or classification suites.

**Description of the datasets** ABALONE<sup>1</sup> Nash et al. [1994] contains several physical measurements on abalone specimens used to predict their age. DELTA-AIRLOINS collects mechanical measurements for the task of controlling the ailerons of a F16 aircraft while the task is to predict the variation of the action on the ailerons. ELEVATORS comprises measurements also concerned with the task of controlling a F16 aircraft (different from DELTA-AIRLOINS), although the target variable here refers on controlling on the elevators of the aircraft. In INSURANCE<sup>2</sup> one wants to predict individual medical costs billed by health insurance given several personal data of a patient. MNIST<sup>3</sup> comprises gray-scale handwritten digit images used for multi-class classification. FASHION<sup>4</sup> is a 10-class image classification challenge concerning fashion apparel items.

**Preprocessing steps** We preserve for all dataset their train and test splits if present in their respective repositories, or create a new test set comprising 20% of the whole data. Moreover we reserve a 10% portion of the training set as validation data used to monitor (parameter and/or structure) learning of our models and perform early-stopping.

We perform discretization of the continuous features in the regression datasets as follows. We first try to automatically detect the optimal number of (irregular) bins through adaptive binning by employing a penalized likelihood scheme as in Rozenholc et al. [2010]. If the number of the bins found in this way exceeds ten, we employ an equal-width binning scheme capping the bin number to ten, instead. Once the data is discrete, we encode them as binary through the common one-hot encoding, to accommodate the requirements of the PSDD learner we employed Liang et al. [2017].

For image data, we binarize each sample by considering each pixel in it to be 1 if its original value exceed the mean value of that pixel as computed on the training set.

Statistics for all the datasets after preprocessing can be found in Table B.1.

---

<sup>1</sup><https://archive.ics.uci.edu/ml/datasets/abalone>

<sup>2</sup><https://www.kaggle.com/mirichoi0218/insurance>

<sup>3</sup><http://yann.lecun.com/exdb/mnist/>

<sup>4</sup><https://github.com/zalandoresearch/fashion-mnist>

Table B.1: Statistics as number of train, validation and test samples and features (after discretization) for the datasets employed in the regression (top half) and classification (bottom half) experiments.

DATASET	TRAIN	VALID	TEST	FEATURES
ABALONE	2923	584	670	71
DELTA-AIRLOINS	4990	998	1141	55
ELEVATORS	11619	2323	2657	182
INSURANCE	936	187	215	36
MNIST	48000	12000	10000	784
FASHION	48000	12000	10000	784

Table B.2: Statistics on the runtime of our algorithm versus MICE and the Monte Carlo Sampling algorithm. The reported times for prediction times are for one configuration of the experiment. As we tried 10 different missingness percentages and repeated each 10 times, the total time of experiment is 100 times the value in the table. Learning time refers to learning the generative circuit and is done only once.

	Time (seconds)				
	ours (learning)	ours (prediction)	MICE	MC	
ABALONE	82		20	43	117
DELTA	53		24	27	126
INSURANCE	40		13	11	20
ELEVATORS	2105		31	364	994

**Runtime of the algorithms** In Table B.2, we report the runtimes for our method versus MICE and expectations approximated via Monte Carlo simulations, by sampling the generative model and evaluating on the discriminative model. As we see, the speed advantage of our algorithm becomes more clear on larger datasets. The runtime of the Monte Carlo approximation depends on number of samples and the size of the generative circuit. The runtime of MICE also depends on missing percentage of features and increases notably as more features go missing. For this reason, and by observing that MICE was providing worse predictions than our algorithm, we stopped MICE experiments early at 30% missing for the ELEVATORS dataset.

## Appendix C

# Handling Missing Data in Decision Trees: A Probabilistic Approach

### C.1 Proofs

**Proposition 4.1** (Expected predictions for decision trees). Given a decision tree  $(\mathcal{T}, \Theta)$  encoding  $f_{\Theta}(\mathbf{x})$ , a distribution  $p(\mathbf{X})$ , and a partial assignment  $\mathbf{x}^o$ , the expected prediction of  $f$  w.r.t.  $p$  can be computed as follows:

$$\mathbb{E}_{\mathbf{x}^m \sim p(\mathbf{X}^m | \mathbf{x}^o)} [f_{\Theta}(\mathbf{x}^o, \mathbf{x}^m)] = \frac{1}{p(\mathbf{x}^o)} \sum_{\ell \in \text{leaves}(\mathcal{T})} \theta_{\ell} \cdot p_{\ell}(\mathbf{x}^o)$$

where  $p_{\ell}(\mathbf{x}^o) = p(\mathbf{x}^{\text{path}(\ell)}, \mathbf{x}^o)$  and  $\mathbf{x}^{\text{path}(\ell)}$  is the assignment to the RVs in  $\text{path}(\ell)$  that evaluates  $\mathcal{I}_{\ell}(\mathbf{x}') = \prod_{(n,j) \in \text{path}(\ell)} \mathbb{1}[x'_n = j]$  to 1.

*Proof.* Let  $\mathbf{X}^{\text{path}(\ell)}$  be the set of RVs appearing in the decision nodes in  $\text{path}(\ell)$ . Then the following holds:

$$\begin{aligned} & \mathbb{E}_{\mathbf{x}^m \sim p(\mathbf{X}^m | \mathbf{x}^o)} [f_{\Theta}(\mathbf{x}^o, \mathbf{x}^m)] \\ &= \frac{1}{p(\mathbf{x}^o)} \mathbb{E}_{\mathbf{x}^m \sim p(\mathbf{X}^m, \mathbf{x}^o)} [f_{\Theta}(\mathbf{x}^o, \mathbf{x}^m)] \\ &= \frac{1}{p(\mathbf{x}^o)} \mathbb{E}_{\mathbf{x}^m \sim p(\mathbf{X}^m, \mathbf{x}^o)} \left[ \sum_{\ell \in \text{leaves}(\mathcal{T})} \theta_{\ell} \mathcal{I}_{\ell}(\mathbf{x}) \right] \end{aligned}$$

$$\begin{aligned}
&= \frac{1}{p(\mathbf{x}^o)} \sum_{\ell \in \text{leaves}(\mathcal{T})} \theta_\ell \mathbb{E}_{\mathbf{x}^m \sim p(\mathbf{X}^m, \mathbf{x}^o)} [\mathcal{I}_\ell(\mathbf{x})] \\
&= \frac{1}{p(\mathbf{x}^o)} \sum_{\ell \in \text{leaves}(\mathcal{T})} \theta_\ell \mathbb{E}_{\mathbf{x}^m \sim p(\mathbf{X}^m, \mathbf{x}^o)} \left[ \prod_{(n,j) \in \text{path}(\ell)} \mathbb{1}[x_n = j] \right] \\
&= \frac{1}{p(\mathbf{x}^o)} \sum_{\ell \in \text{leaves}(\mathcal{T})} \theta_\ell p(\mathbf{x}^{\text{path}(\ell)}, \mathbf{x}^o)
\end{aligned}$$

□

Before proving Proposition 4.2, let us first introduce a useful lemma.

**Lemma 14** (Expected squared predictions). *Given a decision tree structure  $(\mathcal{T}, \Theta)$  encoding  $f_\Theta(\mathbf{x})$ , a distribution  $p(\mathbf{X})$  and a partial assignment  $\mathbf{x}^o$  the expected squared prediction of  $f$  w.r.t.  $p$  can be computed as follows:*

$$\mathbb{E}_{p_\phi(\mathbf{X}^m | \mathbf{x}^o)} [f_\Theta^2(\mathbf{x})] = \frac{1}{p(\mathbf{x}^o)} \sum_{\ell \in \text{leaves}(\mathcal{T})} \theta_\ell^2 p_\ell(\mathbf{x}^o).$$

*Proof.*

$$\begin{aligned}
\mathbb{E}_{p_\phi(\mathbf{X}^m | \mathbf{x}^o)} [f_\Theta^2(\mathbf{x})] &= \frac{1}{p(\mathbf{x}^o)} \mathbb{E}_{p_\phi(\mathbf{X}^m, \mathbf{x}^o)} [f_\Theta^2(\mathbf{x})] \\
&= \frac{1}{p(\mathbf{x}^o)} \mathbb{E}_{p_\phi(\mathbf{X}^m, \mathbf{x}^o)} \left[ \left( \sum_{\ell \in \text{leaves}(\mathcal{T})} \theta_\ell \mathcal{I}_\ell(\mathbf{x}) \right)^2 \right] \\
&= \frac{1}{p(\mathbf{x}^o)} \mathbb{E}_{p_\phi(\mathbf{X}^m, \mathbf{x}^o)} \left[ \sum_{\ell \in \text{leaves}(\mathcal{T})} \theta_\ell^2 \mathcal{I}_\ell(\mathbf{x}) \right] \tag{C.1.1}
\end{aligned}$$

$$\begin{aligned}
&= \frac{1}{p(\mathbf{x}^o)} \sum_{\ell \in \text{leaves}(\mathcal{T})} \theta_\ell^2 \cdot \mathbb{E}_{p_\phi(\mathbf{X}^m, \mathbf{x}^o)} [\mathcal{I}_\ell(\mathbf{x})] \\
&= \frac{1}{p(\mathbf{x}^o)} \sum_{\ell \in \text{leaves}(\mathcal{T})} \theta_\ell^2 \cdot p_\ell(\mathbf{x}^o). \tag{C.1.2}
\end{aligned}$$

where Eq. C.1.1 follows from the fact that  $\mathcal{I}_\ell(\mathbf{x}) \cdot \mathcal{I}_{\ell'}(\mathbf{x}) = 0$  iff  $\ell \neq \ell'$  and from the idempotence of indicator functions ( $\mathcal{I}_j^2(\mathbf{x}) = \mathcal{I}_j(\mathbf{x})$ ), whereas Eq. C.1.2 follows the proof of Proposition 4.1. □

**Proposition 4.2** (Expected parameters of MSE loss). Given a decision tree structure  $\mathcal{T}$  and a training set  $\mathbf{D}_{\text{train}}$ , the set of parameters  $\Theta = \{\theta_\ell\}_{\ell \in \text{leaves}(\mathcal{T})}$  that minimizes  $\mathcal{L}_{\text{MSE}}$ , the expected prediction loss for MSE, can be found by

$$\theta_\ell^* = \frac{\sum_{\mathbf{x}^o, y \in \mathbf{D}_{\text{train}}} y \cdot p_\ell(\mathbf{x}^o)/p(\mathbf{x}^o)}{\sum_{\mathbf{x}^o, y \in \mathbf{D}_{\text{train}}} p_\ell(\mathbf{x}^o)/p(\mathbf{x}^o)}$$

for each leaf  $\ell \in \text{leaves}(\mathcal{T})$ .

*Proof.* First, the expected MSE loss can be expressed as the following:

$$\begin{aligned} \mathcal{L}_{\text{MSE}}(\Theta; \mathbf{D}_{\text{train}}) &= \frac{1}{|\mathbf{D}_{\text{train}}|} \sum_{\mathbf{x}^o, y \in \mathbf{D}_{\text{train}}} \mathbb{E}_{p_\phi(\mathbf{X}^m | \mathbf{x}^o)} [(y - f_\Theta(\mathbf{x}))^2] \\ &= \frac{1}{|\mathbf{D}_{\text{train}}|} \sum_{\mathbf{x}^o, y \in \mathbf{D}_{\text{train}}} \left( y^2 - 2y \mathbb{E}_{p_\phi(\mathbf{X}^m | \mathbf{x}^o)} [f_\Theta(\mathbf{x})] \right. \\ &\quad \left. + \mathbb{E}_{p_\phi(\mathbf{X}^m | \mathbf{x}^o)} [f_\Theta^2(\mathbf{x})] \right). \end{aligned}$$

To optimize this loss, we consider its partial derivative w.r.t. a leaf parameter  $\theta_\ell$ . Using Equation 4.3.1 and the fact that gradient is a linear operator, we have:

$$\frac{\partial \mathbb{E}_{p_\phi(\mathbf{X}^m | \mathbf{x}^o)} [f_\Theta(\mathbf{x})]}{\partial \theta_\ell} = \frac{p_\ell(\mathbf{x}^o)}{p(\mathbf{x}^o)}.$$

Similarly, the partial derivative of expected squared prediction in Lemma 14 is:

$$\frac{\partial \mathbb{E}_{p_\phi(\mathbf{X}^m | \mathbf{x}^o)} [f_\Theta^2(\mathbf{x})]}{\partial \theta_\ell} = \frac{2\theta_\ell p_\ell(\mathbf{x}^o)}{p(\mathbf{x}^o)}.$$

Therefore, the partial derivative of expected MSE loss w.r.t. a leaf parameter  $\theta_\ell$  can be computed

as follows:

$$\begin{aligned}
& \frac{\partial \mathcal{L}_{\text{MSE}}}{\partial \theta_\ell} \\
&= \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{\mathbf{x}^o, y \in \mathcal{D}_{\text{train}}} \left( -2y \frac{\partial \mathbb{E}_{p_\phi(\mathbf{X}^m | \mathbf{x}^o)} [f_\Theta(\mathbf{x})]}{\partial \theta_\ell} \right. \\
&\quad \left. + \frac{\partial \mathbb{E}_{p_\phi(\mathbf{X}^m | \mathbf{x}^o)} [f_\Theta^2(\mathbf{x})]}{\partial \theta_\ell} \right) \\
&= \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{\mathbf{x}^o, y \in \mathcal{D}_{\text{train}}} \left( -2y \frac{p_\ell(\mathbf{x}^o)}{p(\mathbf{x}^o)} + \frac{2\theta_\ell p_\ell(\mathbf{x}^o)}{p(\mathbf{x}^o)} \right).
\end{aligned}$$

Then its gradient w.r.t. the parameter vector  $\boldsymbol{\theta} = [\theta_{\ell_1}, \dots, \theta_{\ell_L}]$ , with  $L = |\text{leaves}(\mathcal{T})|$ , can be written in matrix notation as:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L} = \frac{2}{|\mathcal{D}_{\text{train}}|} \sum_{\mathbf{x}^o, y \in \mathcal{D}_{\text{train}}} \begin{bmatrix} (\theta_{\ell_1} - y) p_{\ell_1}(\mathbf{x}^o)/p(\mathbf{x}^o) \\ \vdots \\ (\theta_{\ell_L} - y) p_{\ell_L}(\mathbf{x}^o)/p(\mathbf{x}^o) \end{bmatrix}$$

Hence, by setting  $\nabla_{\boldsymbol{\theta}} \mathcal{L} = \mathbf{0}$  we can easily retrieve that the optimal parameter vector is:

$$\boldsymbol{\theta}^* = \begin{bmatrix} \frac{\sum_{\mathbf{x}^o, y \in \mathcal{D}_{\text{train}}} y \cdot p_{\ell_1}(\mathbf{x}^o)/p(\mathbf{x}^o)}{\sum_{\mathbf{x}^o, y \in \mathcal{D}_{\text{train}}} p_{\ell_1}(\mathbf{x}^o)/p(\mathbf{x}^o)} \\ \vdots \\ \frac{\sum_{\mathbf{x}^o, y \in \mathcal{D}_{\text{train}}} y \cdot p_{\ell_L}(\mathbf{x}^o)/p(\mathbf{x}^o)}{\sum_{\mathbf{x}^o, y \in \mathcal{D}_{\text{train}}} p_{\ell_L}(\mathbf{x}^o)/p(\mathbf{x}^o)} \end{bmatrix}$$

**Regularization.** During parameter learning, it is common to also add a regularization term to the total loss to reduce overfitting. In our case, we use regularizer  $L_2(\Theta) = \|\Theta\|^2 = \sum_i \theta_i^2$ . Now, we want to minimize the following loss:

$$\mathcal{L} = \mathcal{L}_{\text{MSE}} + \lambda L_2 \tag{C.1.3}$$

Where  $\lambda$  is the regularization hyperparameter. By repeating the steps from above we can easily see that the parameters that minimize  $\mathcal{L}$  are:

$$\boldsymbol{\theta}^* = \begin{bmatrix} \frac{\sum_{\mathbf{x}^o, y \in D_{\text{train}}} y \cdot p_{\ell_1}(\mathbf{x}^o)/p(\mathbf{x}^o)}{\lambda + \sum_{\mathbf{x}^o, y \in D_{\text{train}}} p_{\ell_1}(\mathbf{x}^o)/p(\mathbf{x}^o)} \\ \vdots \\ \frac{\sum_{\mathbf{x}^o, y \in D_{\text{train}}} y \cdot p_{\ell_L}(\mathbf{x}^o)/p(\mathbf{x}^o)}{\lambda + \sum_{\mathbf{x}^o, y \in D_{\text{train}}} p_{\ell_L}(\mathbf{x}^o)/p(\mathbf{x}^o)} \end{bmatrix}$$

□

## C.2 Expected Parameters Beyond Single Trees

In this section, we extend the expected parameter tuning to beyond single tree models. The learning scenarios include forests, bagging, random forests, and gradient tree boosting.

### C.2.1 Forests

In this section, instead of a single tree  $f_\theta$ , we are given a forest  $F_\Theta$ , and want to minimize the following loss instead:

$$\mathcal{L}_{\text{Forest}}(\Theta; D) = \frac{1}{|D|} \sum_{\mathbf{x}^o, y \in D} \mathbb{E}_{p_\Phi(\mathbf{X}^m | \mathbf{x}^o)} [l(y, F_\Theta(\mathbf{x}))]$$

**Proposition C.1** (Expected parameters of forests MSE loss). *Given the training set  $D_{\text{train}}$ , and given the Forest  $F_\theta$ , the set of parameters  $\Theta$  that minimizes  $\mathcal{L}_{\text{Forest}}$  can be found by solving for  $\Theta$  in the following linear system of equations:*

$$M \times \Theta = B \tag{C.2.1}$$



where  $M$  is  $k \times k$  matrix,  $\Theta$  and  $B$  are  $k \times 1$  vectors.

$$\begin{aligned}
 M[i, j] &= \sum_{\mathbf{x}^o, y \in \mathcal{D}_{\text{train}}} p_{\ell_i, \ell_j}(\mathbf{x}^o) / p(\mathbf{x}^o) \\
 B[i] &= \sum_{\mathbf{x}^o, y \in \mathcal{D}_{\text{train}}} y \cdot p_{\ell_i}(\mathbf{x}^o) / p(\mathbf{x}^o) \\
 \Theta[i] &= \theta_i
 \end{aligned}$$

Note that, we usually learn forest, tree by tree and do not have all the tree structures initially, and also above algorithm grows quadratic to number of total leaves which is less desirable. As a result, we also want to explore other scenarios such as bagging or boosting.

### C.2.2 Bagging and Random Forests

In both Bagging of trees and Random forests, we learn our trees independently and average their predictions, we can also do the expected parameter tuning for each tree independently, w.r.t. a generative model learned on the bootstrap sample of the training dataset on which the tree has been induced.

### C.2.3 Gradient Tree Boosting

In this section, we adapt gradient tree boosting in the expected prediction framework. Before moving on to boosting of trees, we introduce Lemma 15, which computes the expected prediction of two trees multiplied.

**Lemma 15** (Expected tree times tree). *Given two trees  $f_{\Theta}(\mathbf{x})$ , and  $f'_{\Theta}(\mathbf{x})$ , a distribution  $p(\mathbf{X})$  and a partial assignment  $\mathbf{x}^o$  the expected squared prediction of  $f(\mathbf{x}) \cdot f'(\mathbf{x})$  w.r.t.  $p$  can be computed as*

follows:

$$\mathbb{E}_{p_\Phi(\mathbf{X}^m|\mathbf{x}^o)} [f_\theta \cdot f_{\theta'}] = \frac{\sum_{\ell \in \text{leaves}(f)} \sum_{j \in \text{leaves}(f')} \theta_\ell \theta_j p_{\ell,j}(\mathbf{x}^o)}{p(\mathbf{x}^o)}$$

where  $p_{\ell,j}(\mathbf{x}^o) = p(\mathbf{x}^{\text{path}(\ell)}, \mathbf{x}^{\text{path}(j)}, \mathbf{x}^o)$ .

*Proof.* The proof similarly follows from proof of Lemma 14. The main difference is that in  $\mathcal{I}_\ell(\mathbf{x}) \cdot \mathcal{I}_{\ell'}(\mathbf{x})$  the leaves  $\ell$  and  $\ell'$  are from two different trees so its not necessarily equal to 0, so we can not cancel those terms.  $\square$

Note that Lemma 15 result can be easily extended to multiplying two forests.

During gradient tree boosting we learn our forest in a additive manner. At each step, given the already learned forest  $F$  we add a new tree  $f_\theta$  that minimizes sum of losses of the form  $l(y, F(\mathbf{x}) + f_\theta(\mathbf{x}))$ . We adapt this with the expected prediction framework as follows:

**Definition 24** (Boosting expected loss minimization). *In addition to definition 7, we are also given a fixed forest  $F$ , we want to find the set of parameters  $\Theta$  of the tree such that  $f_\Theta$  minimizes the expected loss  $\mathcal{L}_{Boost}$  defined as follows:*

$$\mathcal{L}_{Boost}(\Theta; D) = \frac{1}{|D|} \sum_{\mathbf{x}^o, y \in D} \mathbb{E}_{p_\Phi(\mathbf{X}^m|\mathbf{x}^o)} [l(y, F(\mathbf{x}) + f_\Theta(\mathbf{x}))]$$

**Proposition C.2** (Expected parameters of Boosted MSE loss). *Given the training set  $D_{\text{train}}$ , and given the Forest  $F$  and the new tree structures  $f_\Theta$ , the set of parameters  $\Theta$  that minimizes  $\mathcal{L}_{Boost}$  can be found by*

$$\theta_\ell^* = \frac{\sum_{\mathbf{x}^o, y \in D_{\text{train}}} \frac{y \cdot p_\ell(\mathbf{x}^o) - \sum_{j \in \text{leaves}(F)} \theta_j p_{\ell,j}(\mathbf{x}^o)}{p(\mathbf{x}^o)}}{\sum_{\mathbf{x}^o, y \in D_{\text{train}}} \frac{p_\ell(\mathbf{x}^o)}{p(\mathbf{x}^o)}}$$

Table C.1: Statistics about the datasets used in the experiments.

DATASET	TRAIN	VALID	TEST	FEATURES
INSURANCE	936	187	215	36

### C.3 More Experiment Info

**Description of the datasets** In the INSURANCE <sup>1</sup> dataset, the goal is to predict yearly medical insurance costs of patients given other attributes such as age, gender, and whether they smoke or not.

**Preprocessing Steps** We preserve the original test, and train splits if present for each dataset. Additionally, we merge any given validation set with the test set.

The probabilistic circuit learning implementation that we use does not support continuous features yet, so we perform discretization of the continuous features as follows. First, we try to automatically detect the optimal number of (irregular) bins through adaptive binning by employing a penalized likelihood scheme as in Rozenholc et al. [2010]. If the number of the bins found in this way exceeds ten, instead we employ an equal-width binning scheme capping the bin number to ten. Once the data is discrete, we employ one-hot encoding.

**Other Settings** For XGBoost, we use “reg:squarederror” which corresponds to MSE loss. Max depth is set to 5, and we use regularization  $\lambda = 1$  where applicable.

When learning XGBoost trees from missing values, some of the leaves become only reachable if a certain feature is missing, and never reachable with fully observed data. We ignore those leaves in our expected prediction framework.

---

<sup>1</sup><https://www.kaggle.com/mirichoi0218/insurance>

# Appendix D

## Probabilistic Sufficient Explanations

### D.1 Proof of Theorem 7

*Proof.* We provide the proof for the case of using  $\text{EP} = \text{EP}_f$ , and  $U(\mathbf{z})$  providing an upper bound on the probabilistic predictor  $f(\mathbf{z}\mathbf{M})$ . The proof is mostly identical in the case of having  $\text{EP} = \text{EP}_{\mathcal{O}}$  and  $U(\mathbf{z})$  being an upper bound for the log-odds predictor  $\mathcal{O}(\mathbf{z}\mathbf{M})$ .

Let  $T$  be the decision threshold. Then we have  $\text{SDP } \mathbf{xz} = \Pr(f(\mathbf{z}\mathbf{m}) \geq T)$  and  $\text{EP}_f(\mathbf{z}) = \mathbb{E}[f(\mathbf{z}\mathbf{m})]$  where  $\mathbf{m} \sim \Pr(\mathbf{M}|\mathbf{z})$ . Thus,

$$\begin{aligned} \text{EP}_f(\mathbf{z}) &= \mathbb{E}[f(\mathbf{z}\mathbf{m})] \\ &= \mathbb{E}[f(\mathbf{z}\mathbf{m})|f(\mathbf{z}\mathbf{m}) < T] \cdot \Pr(f(\mathbf{z}\mathbf{m}) < T) \\ &\quad + \mathbb{E}[f(\mathbf{z}\mathbf{m})|f(\mathbf{z}\mathbf{m}) \geq T] \cdot \Pr(f(\mathbf{z}\mathbf{m}) \geq T) \\ &< T(1 - \Pr(f(\mathbf{z}\mathbf{m}) \geq T)) \\ &\quad + U(\mathbf{z}) \Pr(f(\mathbf{z}\mathbf{m}) \geq T) \\ &= T + (U(\mathbf{z}) - T) \Pr(f(\mathbf{z}\mathbf{m}) \geq T) \\ &= T + (U(\mathbf{z}) - T) \text{SDP } \mathbf{xz}. \end{aligned}$$

Rearranging the terms leads to Equation 5.4.1.

Next we show how to construct a distribution for  $f(\mathbf{z}\mathbf{M})$  to demonstrate the tightness of the bound. Assume we are given  $\text{EP}_f(\mathbf{z}) = F$ ,  $U(\mathbf{z}) = U$ , and a decision threshold  $T$  with  $F < T < U$ .

Then let  $\epsilon > 0$  and consider the distribution given by

$$P(f(\mathbf{z}\mathbf{M}) = k) = \begin{cases} \frac{F - T}{U - T} + \epsilon & \text{if } k = U \\ \frac{U - F}{U - T} - \epsilon & \text{if } k = A \\ 0 & \text{otherwise} \end{cases}$$

where  $A$  is some constant which makes  $\text{EP}_f(\mathbf{z}) = F$  and  $A < T$  so that  $\text{SDP } \mathbf{xz} = \frac{F-T}{U-T} + \epsilon$ . We now show how to solve for  $A$ .

We start by computing expected prediction (Definition 9), we have:

$$\begin{aligned} \text{EP}_f(\mathbf{z}) &= \mathbb{E}_{\mathbf{m} \sim \text{Pr}(\mathbf{M}|\mathbf{z})} f(\mathbf{z}\mathbf{m}) \\ &= U \times P(f(\mathbf{z}\mathbf{M}) = U) + A \times P(f(\mathbf{z}\mathbf{M}) = A) + 0 \\ &= U \left( \frac{F - T}{U - T} + \epsilon \right) + A \left( \frac{U - F}{U - T} - \epsilon \right) \\ &= F \end{aligned}$$

and then solve for  $A$ :

$$\begin{aligned} A &= \frac{F - U \left( \frac{F-T}{U-T} + \epsilon \right)}{\left( \frac{U-F}{U-T} - \epsilon \right)} \\ &= \frac{F(U - T) - U(F - T) - U(U - T)\epsilon}{U - F - (U - T)\epsilon} \\ &= \frac{T(U - F) - U(U - T)\epsilon}{U - F - (U - T)\epsilon} \\ &< \frac{T(U - F) - T(U - T)\epsilon}{U - F - (U - T)\epsilon} \\ &= T \end{aligned}$$

Thus we have  $\text{EP}_f(\mathbf{z}) = F$  and, since  $A < T$ ,  $\text{SDP } \mathbf{xz} = \frac{F-T}{U-T} + \epsilon$ . □

## D.2 Beam Search Algorithm

### D.2.1 Pseudocode

In the above pseudocode, the `expand` function generates a new set of candidates, each formed by adding one yet unobserved feature to the input candidate. The `top_b` function selects the  $b$  candidates with the highest EP, with ties broken using `Pr`. Finally, the MLSE is updated if an explanation with a higher EP is found, or if one has the same EP as the previous MLSE but a higher `Pr`.

### D.2.2 Computational Complexity

When using PCs to model the feature distribution, the runtimes for tractable expected prediction (EP) and marginal (`Pr`) algorithms do not depend on how many features are observed. `Pr` can be computed in linear time w.r.t. circuit size. Linearity of decision forest classification also allows for expected prediction computations in time linear w.r.t. circuit size. Thus we treat EP and `Pr` as oracles and measure our algorithm’s runtime by the number of calls to them. In each level of the beam search, we expand the previous level’s top  $b$  candidates, giving us at most  $nb$  unique states, where  $n$  is the number of features. We then call EP and `Pr` once for each state. Since the search stops after level  $k$ , overall we make in total  $O(nkb)$  calls to the EP and `Pr` oracles.

## D.3 More Experiment Details

### D.3.1 Computing Infrastructure

All experiments were run on a Linux server with 40 CPU cores and 500 GB of RAM, albeit not all the memory or CPU cores were needed for generating one explanation. Our method is highly parallelizable as we see adding more threads speeds up the process substantially. No GPUs were used in this paper.

Although we did not use any GPUs, it should be easy to re-implement some parts of our method

to support GPUs. Some of the main ingredients of our method such as computing marginal probabilities using Probabilistic Circuits already support GPUs and enjoy huge speed ups. Furthermore, computing expected predictions and the search at each level should be amenable to GPU parallelization. So, overall we should be able to speed up the algorithm by 1-2 orders of magnitude. This allows us to scale to even more complicated models with more features in the future.

### **D.3.2 Datasets and Preprocessing Steps**

The MNIST Yann et al. [2009] dataset consists of 60,000 grayscale images ( $28 \times 28$  pixels) of handwritten digits (0-9). We limit the dataset into digits of 3 and 5 to ensure we have a binary classification task. The dataset is already split between train and test images, so we used the same split, selecting from each digits of 3s and 5s. Since our probabilistic circuit library currently only supports binary feature values, we also binarized pixel values. independently for each image, by applying a threshold at 0.05 standard deviations above the mean.

The classification task for the adult dataset Kohavi [1996] is to determine whether a given individual makes over \$50,000 per year. Features include age, sex, working class, hours worked per week, education level, nationality, etc. We perform discretization of continuous features by applying standard binning. We then perform one-hot-encoding for all features. Again, this is because the probabilistic circuit library only supports binary features. We then used an 80-20 split for our training and test data. In general, our method should be straightforward to extend to multi-category features and also continuous features.

Source code for processing the datasets will be included in the final version of paper as supplementary.

### **D.3.3 Details on models**

For our MNIST experiments, we used a PC with 10124 nodes, 15640 edges, and 5916 parameters. It took approximately 35 minutes to train, stopping after 410 iterations. For our classifier, we trained

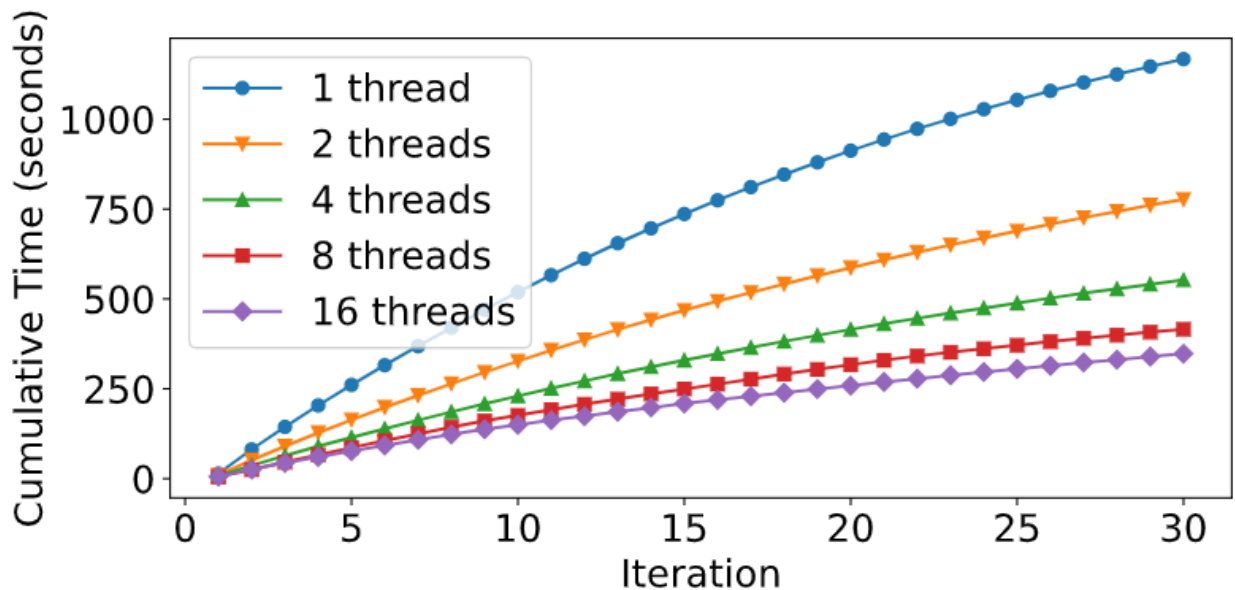


Figure D.1: MNIST Runtimes. Average cumulative time taken until completion of each iteration of the beam search algorithm for different numbers of threads.

a decision forest with 6 trees, each with a maximum depth of 6. The final model has a total of 297 leaves. The classification accuracy on our test images is 98.63%.

For our adult experiments, we used a PC with 22899 nodes, 44152 edges, and 14828 parameters. It took approximately 17 minutes to train, stopping after 190 iterations of structure learning. For our classifier, we trained a decision forest with the same constraints as above. The final model has a total of 255 leaves. The classification accuracy on our test dataset is 84.20%.

### D.3.4 Metric Calculation Details

The classifiers used in our experiments were decision forests, which allow for efficient computation of expected log-odds. Classification using these models is done by summing the weights of one leaf from each tree, where the chosen leaf from each tree has a path from the root not contradicting with the instance to be classified. This additive model allows us to utilize linearity of expectation to compute expected log-odds Khosravi et al. [2020].



This model does not, however, allow for efficient computation of expected predictions, where prediction refers to taking the sigmoid of the classifier output. This is due to the non-linear nature of the sigmoid function. Instead, when we give expected prediction values, we use a first order approximation by taking the sigmoid of the expected log-odds.

In Figure 5.2 we show a plot with SDP lower bounds computed using Theorem 7. Graphing of the blue curve requires computation of an upper bound  $U(\mathbf{z})$  on the predictions of a decision forest. The upper bound used is a loose one which we computed by adding the weights of one leaf from each tree in the forest, where each leaf is the maximum weighted leaf in its tree whose path from the root does not contradict with the explanation. This bound is a loose one since paths for leaves from different trees may contradict. For the red curve, since our model outputs log-odds, we used the first order approximation mentioned above.

### D.3.5 Effects of Parallelism on Performance

One advantage of our method is that we can parallelize computation of expected predictions at each level of the beam search. The effects of can be seen in Figure D.1, which plots the cumulative time taken to complete each level of the beam search for different numbers of threads. We see that more threads do indeed allow for much faster runtimes. We leave investigating benefits of GPU acceleration for future work.

### D.3.6 More on SDP Lowerbounds

In the main text we saw that Theorem 7 gives us two ways of computing a lower bound for SDP: using a probabilistic predictor  $f$  or a log-odds predictor  $\mathcal{O}$ . One of them might give a tighter bound, depending on how our predictor is defined. Generally, the tighter bound we can find for  $U(\mathbf{z})$  the tighter bound we get for SDP.

For example, in the case of having a probabilistic predictor, the decision threshold is usually  $T = 0.5$ , and we can easily get a trivial upper bound of  $U(\mathbf{z}) = 1$ . Depending on the model family

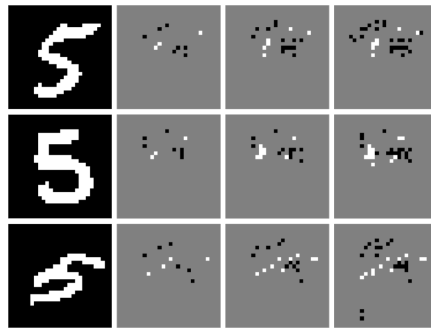
we might be able to get a tighter bound for  $U(\mathbf{z})$ , and hence a tighter lower bound for SDP. For the case of a log-odds predictor, we usually take  $T = 0$ , but getting an upper-bound  $U(\mathbf{z})$  might not be trivial. However, in many cases an upper bound  $U(\mathbf{z})$  can be computed efficiently for log-odds predictor, but might not be tight.

### **D.3.7 Limitations**

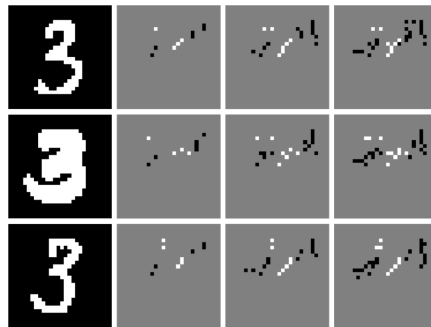
Some knowledge of probabilities is needed to fully interpret the results. However, as seen with the MNIST example, visual representations can be designed to give users a feel for what the model is doing.

### **D.3.8 More MNIST examples**

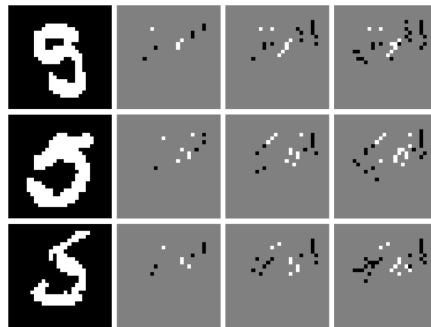
Figure D.2 provides some extra examples on sufficient explanations for MNIST. We see examples for both correctly and incorrectly classified images, and see the effects of cardinality constraints on the explanations.



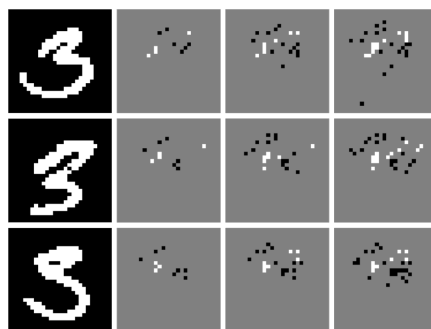
(a) Correctly classified 5s



(b) Correctly classified 3s



(c) Misclassified 5s



(d) Misclassified 3s

Figure D.2: More MNIST examples along with explanations with cardinality constraint  $k = 10, 20, 30$ .

## Appendix E

# Why Is This an Outlier? Explaining Outliers by Submodular Optimization of Marginal Distributions

### Other Search Scenarios

#### Unknown Number of Outlier Features

In most cases, we do not know beforehand how many features we need to provide a explanation for the outlier. One option is to choose a fixed budget  $k$ , and choose the best subset of size  $k$  we can find. However, in some cases, choosing features might have also have a “cost” so we want to have some penalty for choosing more features. One way to handle that would be adding regularization terms based on  $|S|$  to the search. For example:

$$\mathbf{x}_{out} = \arg \max_{|S| \leq k} 1 - p_{\theta}(\mathbf{x}_S) - \lambda \frac{1}{2^{|S|}}. \quad (\text{E.0.1})$$

The upside is that  $-\lambda \frac{1}{2^{|S|}}$  is submodular, so we still get the guarantees from greedy search for submodular optimization. One downside is the need for choosing the hyperparameter  $\lambda$ .

## Bibliography

- Edgar Acuña and Caroline Rodriguez. On detection of outliers and their effect in supervised classification. *University of Puerto Rico at Mayaguez*, 15, 2004.
- Kareem Ahmed, Zhe Zeng, Mathias Niepert, and Guy Van den Broeck. Simple: A gradient estimator for k-subset sampling, , 2023.
- Melissa J Azur, Elizabeth A Stuart, Constantine Frangakis, and Philip J Leaf. Multiple imputation by chained equations: what is it and how does it work? *International journal of methods in psychiatric research*, 2011.
- Gustavo EAPA Batista and Maria Carolina Monard. An analysis of four missing data treatment methods for supervised learning. *Applied artificial intelligence*, 17(5-6):519–533, 2003.
- Gustavo EAPA Batista, Maria Carolina Monard, et al. A study of k-nearest neighbour as an imputation method. *HIS*, 2002.
- Christopher M Bishop. Novelty detection and neural network validation. *IEE Proceedings-Vision, Image and Signal processing*, 141(4):217–222, 1994.
- Jock A Blackard and Denis J Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and electronics in agriculture*, 1999.
- Guillaume Bouchard and Bill Triggs. The tradeoff between generative and discriminative classifiers. In *16th IASC International Symposium on Computational Statistics (COMPSTAT'04)*, pages 721–728, 2004.
- Azzedine Boukerche, Lining Zheng, and Omar Alfandi. Outlier detection: Methods, models, and classification. *ACM Comput. Surv.*, jun 2020. ISSN 0360-0300. doi: 10.1145/3381028. URL <https://doi.org/10.1145/3381028>.

- Stephen Boyd, Seung-Jean Kim, Lieven Vandenbergh, and Arash Hassibi. A tutorial on geometric programming. *Optimization and engineering*, 8(1):67, 2007.
- Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.
- Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.
- Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.
- Tom B. Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. Adversarial patch, 2017. URL <https://arxiv.org/abs/1712.09665>.
- S van Buuren and Karin Groothuis-Oudshoorn. mice: Multivariate imputation by chained equations in r. *Journal of statistical software*, pages 1–68, 2010.
- Stef van Buuren. *Flexible imputation of missing data*. CRC Press, 2018.
- Oana-Maria Camburu, Eleonora Giunchiglia, Jakob Foerster, Thomas Lukasiewicz, and Phil Blunsom. The struggles of feature-based explanations: Shapley values vs. minimal sufficient subsets. *arXiv*, 2020.
- Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, July 2009. ISSN 0360-0300.
- Chun-Hao Chang, Elliot Creager, Anna Goldenberg, and David Duvenaud. Explaining image classifiers by counterfactual generation. In *ICLR*, 2019.
- Jianbo Chen, Le Song, Martin Wainwright, and Michael Jordan. Learning to explain: An information-theoretic perspective on model interpretation. In *ICML*, 2018.
- Suming Chen, Arthur Choi, and Adnan Darwiche. The same-decision probability: A new tool for decision making. In *PGM*, 2012.

- Suming Jeremiah Chen, Arthur Choi, and Adnan Darwiche. An exact algorithm for computing the same-decision probability. In *IJCAI*, 2013.
- Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16*. Association for Computing Machinery, 2016.
- Ping-yeh Chiang\*, Renkun Ni\*, Ahmed Abdelkader, Chen Zhu, Christoph Studor, and Tom Goldstein. Certified defenses for adversarial patches. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=HyeaSkryYPH>.
- Arthur Choi, Yexiang Xue, and Adnan Darwiche. Same-decision probability: A confidence measure for threshold-based decisions. *International Journal of Approximate Reasoning*, 53(9):1415–1428, 2012.
- Arthur Choi, Guy Van Den Broeck, and Adnan Darwiche. Tractable learning for structured probability spaces: A case study in learning preference distributions. In *Proceedings of the 24th International Conference on Artificial Intelligence, IJCAI'15*, pages 2861–2868. AAAI Press, 2015a. ISBN 978-1-57735-738-4. URL <http://dl.acm.org/citation.cfm?id=2832581.2832649>.
- Arthur Choi, Guy Van den Broeck, and Adnan Darwiche. Tractable learning for structured probability spaces: A case study in learning preference distributions. In *Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI)*, 2015b.
- YooJung Choi and Guy Van den Broeck. On robust trimming of bayesian network classifiers. In *IJCAI*, 2018.
- YooJung Choi, Adnan Darwiche, and Guy Van den Broeck. Optimal feature selection for decision robustness in bayesian networks. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*, August 2017a. doi: 10.24963/ijcai.2017/215.

- YooJung Choi, Adnan Darwiche, and Guy Van den Broeck. Optimal feature selection for decision robustness in bayesian networks. *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*, Aug 2017b.
- YooJung Choi, Golnoosh Farnadi, Behrouz Babaki, and Guy Van den Broeck. Learning fair naive bayes classifiers by discovering and eliminating discrimination patterns. *CoRR*, abs/1906.03843, 2019.
- YooJung Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic models, oct 2020a. URL <http://starai.cs.ucla.edu/papers/ProbCirc20.pdf>.
- YooJung Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic models, 2020b.
- Xu Chu, Ihab F Ilyas, Sanjay Krishnan, and Jiannan Wang. Data cleaning: Overview and emerging challenges. In *Proceedings of the 2016 international conference on management of data*, pages 2201–2206, 2016.
- Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. Emnist: an extension of mnist to handwritten letters, 2017. URL <https://arxiv.org/abs/1702.05373>.
- Adriana Fonseca Costa and et al. Missing data imputation via denoising autoencoders: the untold story. In *Int’l Symposium on Intelligent Data Analysis*, 2018.
- Meihua Dang, Pasha Khosravi, Yitao Liang, Antonio Vergari, and Guy Van den Broeck. Juice: A julia package for logic and probabilistic circuits. In *AAAI Conference (Demo Track)*, 2021. URL <http://starai.cs.ucla.edu/papers/DangAAAI21.pdf>.
- Meihua Dang, Antonio Vergari, and Guy Van den Broeck. Strudel: A fast and accurate learner of structured-decomposable probabilistic circuits. *International Journal of Approximate Reasoning*, 140:92–115, 2022.



- Adnan Darwiche. A differential approach to inference in bayesian networks. *J.ACM*, 2003.
- Adnan Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009a. doi: 10.1017/CBO9780511811357.
- Adnan Darwiche. *Modeling and reasoning with Bayesian networks*. Cambridge University Press, 2009b.
- Adnan Darwiche and Auguste Hirth. On the reasons behind decisions. In *European Conference on Artificial Intelligence (ECAI)*, 2020.
- Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.
- Rina Dechter. Reasoning with probabilistic and deterministic graphical models: Exact algorithms. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 7(3):1–191, 2013.
- Ofer Dekel and Ohad Shamir. Learning to classify with missing and corrupted features. In *ICML*, 2008.
- Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 1977.
- Laurens Devos, Wannes Meert, and Jesse Davis. Fast gradient boosting decision trees with bit-level data structures. *Proceedings of ECML PKDD, Springer*, 2019.
- Laurens Devos, Wannes Meert, and Jesse Davis. Additive tree ensembles: Reasoning about potential instances. *arXiv*, 2020.
- Botty Dimanov, Umang Bhatt, Mateja Jamnik, and Adrian Weller. You shouldn't trust me: Learning models which conceal unfairness from multiple explanation methods. In *ECAI*, 2020.
- Dheeru Dua and Efi Karra Taniskidou. UCI machine learning repository, 2017. URL <http://archive.ics.uci.edu/ml>.

- Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.
- Laura Isabel Galindez Olascoaga, Wannes Meert, Marian Verhelst, and Guy Van den Broeck. Towards hardware-aware tractable learning of probabilistic models. In *Advances in Neural Information Processing Systems 32 (NeurIPS)*, 2019.
- Sachin Gavankar and Sudhirkumar Sawarkar. Decision tree: Review of techniques for missing values at training, testing and compatibility. In *2015 3rd International Conference on Artificial Intelligence, Modelling and Simulation (AIMS)*, pages 122–126. IEEE, 2015.
- Amir Globerson and Sam Roweis. Nightmare at test time: Robust learning by feature deletion. In *ICML*, 2006.
- Lovedeep Gondara and Ke Wang. Mida: Multiple imputation using denoising autoencoders. In *SIGKDD*, 2018.
- John W Graham. *Missing data: Analysis and design*. Springer Science & Business Media, 2012.
- Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM Comput. Surv.*, 2018.
- Victoria Hodge and Jim Austin. A survey of outlier detection methodologies. *Artificial intelligence review*, 22(2):85–126, 2004.
- Alexey Ignatiev. Towards Trustable Explainable AI. In *IJCAI*, 7 2020.
- Alexey Ignatiev, Nina Narodytska, and Joao Marques-Silva. Abduction-based explanations for machine learning models. *Proceedings of the AAAI Conference*, 2019a.
- Alexey Ignatiev, Nina Narodytska, and Joao Marques-Silva. On validating, repairing and refining heuristic ml explanations, 2019b.

- Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features. *Advances in neural information processing systems*, 32, 2019.
- Rishabh Iyer, Ninad Khargoankar, Jeff Bilmes, and Himanshu Asanani. Submodular combinatorial information measures with applications in machine learning. In Vitaly Feldman, Katrina Ligett, and Sivan Sabato, editors, *Proceedings of the 32nd International Conference on Algorithmic Learning Theory*, volume 132 of *Proceedings of Machine Learning Research*. PMLR, 2021. URL <https://proceedings.mlr.press/v132/iyer21a.html>.
- Yacine Izza, Alexey Ignatiev, and Joao Marques-Silva. On explaining decision trees. *arXiv*, 2020.
- Tommi Jaakkola and David Haussler. Exploiting generative models in discriminative classifiers. In *Advances in neural information processing systems*, 1999.
- Julie Josse, Nicolas Prost, Erwan Scornet, and Gaël Varoquaux. On the consistency of supervised learning with missing values. *arXiv preprint arXiv:1902.06931*, 2019.
- Jihed Khiari, Luis Moreira-Matias, Ammar Shaker, Bernard Ženko, and Sašo Džeroski. Metabags: Bagged meta-decision trees for regression. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 637–652. Springer, 2018.
- Pasha Khosravi, YooJung Choi, Yitao Liang, Antonio Vergari, and Guy Van den Broeck. On tractable computation of expected predictions. In *Advances in Neural Information Processing Systems*, 2019a.
- Pasha Khosravi, Yitao Liang, YooJung Choi, and Guy Van den Broeck. What to expect of classifiers? Reasoning about logistic regression with missing features. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, 2019b.
- Pasha Khosravi, Yitao Liang, YooJung. Choi, and Guy Van den Broeck. What to expect of classifiers? reasoning about logistic regression with missing features. In *IJCAI*, 2019c.

- Pasha Khosravi, Antonio Vergari, YooJung Choi, Yitao Liang, and Guy Van den Broeck. Handling missing data in decision trees: A probabilistic approach. In *Proceedings of The Art of Learning with Missing Values, Workshop at ICML, 2020*.
- Doga Kisa, Guy Van den Broeck, Arthur Choi, and Adnan Darwiche. Probabilistic sentential decision diagrams. In *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, July 2014.
- Ron Kohavi. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *KDD*, 1996.
- Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.
- Andreas Krause and Daniel Golovin. Submodular function maximization. *Tractability*, 3:71–104, 2014.
- Andreas Krause and Carlos Guestrin. Optimal value of information in graphical models. *Journal of Artificial Intelligence Research*, 35:557–591, 2009.
- Anita Krishnakumar. Active learning literature survey. *Tech. rep., Technical reports, University of California, Santa Cruz.*, 42, 2007.
- Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998. ISSN 0018-9219. doi: 10.1109/5.726791.
- Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '07*.

Association for Computing Machinery, 2007. ISBN 9781595936097. URL <https://doi.org/10.1145/1281192.1281239>.

Alexander Levine and Soheil Feizi. (de)randomized smoothing for certifiable defense against patch attacks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6465–6475. Curran Associates, Inc., 2020. URL <https://proceedings.neurips.cc/paper/2020/file/47ce0875420b2dbacfc5535f94e68433-Paper.pdf>.

Wenzhe Li, Zhe Zeng, Antonio Vergari, and Guy Van den Broeck. Tractable computation of expected kernels. In *Proceedings of the 37th Conference on Uncertainty in Artificial Intelligence (UAI)*, 2021.

Yitao Liang and Guy Van den Broeck. Towards compact interpretable models: Shrinking of learned probabilistic sentential decision diagrams. In *IJCAI 2017 Workshop on Explainable Artificial Intelligence (XAI)*, 2017.

Yitao Liang and Guy Van den Broeck. Learning logistic circuits. In *Proceedings of the 33rd Conference on Artificial Intelligence (AAAI)*, 2019a.

Yitao Liang and Guy Van den Broeck. Learning logistic circuits. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence*, 2019b.

Yitao Liang, Jessa Bekker, and Guy Van den Broeck. Learning the structure of probabilistic sentential decision diagrams. In *Proceedings of the 33rd Conference on Uncertainty in Artificial Intelligence (UAI)*, 2017.

Roderick JA Little and Donald B Rubin. *Statistical analysis with missing data*, 2014.

Roderick JA Little and Donald B Rubin. *Statistical analysis with missing data*, volume 793. Wiley, 2019.

- Anji Liu and Guy Van den Broeck. Tractable regularization of probabilistic circuits. In *Advances in Neural Information Processing Systems 35 (NeurIPS)*, dec 2021a. URL <http://starai.cs.ucla.edu/papers/LiuNeurIPS21.pdf>.
- Anji Liu and Guy Van den Broeck. Tractable regularization of probabilistic circuits. *Advances in Neural Information Processing Systems*, 34, 2021b.
- Anji Liu, Stephan Mandt, and Guy Van den Broeck. Lossless compression with probabilistic circuits. *International Conference of Learning Representations*, 2022.
- Hancong Liu, Sirish Shah, and Wei Jiang. On-line outlier detection and data cleaning. *Computers & chemical engineering*, 28(9):1635–1647, 2004.
- Ninghao Liu, Donghwa Shin, and Xia Hu. Contextual outlier interpretation. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 2461–2467. International Joint Conferences on Artificial Intelligence Organization, 7 2018. doi: 10.24963/ijcai.2018/341. URL <https://doi.org/10.24963/ijcai.2018/341>.
- Yajing Liu, Edwin KP Chong, Ali Pezeshki, and Zhenliang Zhang. Submodular optimization problems and greedy strategies: A survey. *Discrete Event Dynamic Systems*, 30(3):381–412, 2020.
- Stefan Lüdtkke, Christian Bartelt, and Heiner Stuckenschmidt. Outlying aspect mining via sum-product networks. In Hisashi Kashima, Tsuyoshi Ide, and Wen-Chih Peng, editors, *Advances in Knowledge Discovery and Data Mining*, pages 27–38, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-33374-3.
- Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017a.

Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions, 2017b.

Joao Marques-Silva, Thomas Gerspacher, Martin C. Cooper, Alexey Ignatiev, and Nina Narodytska. Explaining naive bayes and other linear classifiers with polynomial time and delay. In *NeurIPS*, 2020.

Pierre-Alexandre Mattei and Jes Frellsen. MIWAE: Deep generative modelling and imputation of incomplete data sets. In *ICML*, 2019.

Patrick E McKnight, Katherine M McKnight, Souraya Sidani, and Aurelio Jose Figueredo. *Missing data: A gentle introduction*. Guilford Press, 2007.

Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, Amin Karbasi, Jan Vondrak, and Andreas Krause. Lazier than lazy greedy, 2014.

Kevin P Murphy. *Probabilistic machine learning: an introduction*. MIT press, 2022.

Eric Nalisnick, Akihiro Matsukawa, Yee Whye Teh, Dilan Gorur, and Balaji Lakshminarayanan. Do deep generative models know what they don't know? *arXiv preprint arXiv:1810.09136*, 2018.

Warwick J Nash, Tracy L Sellers, Simon R Talbot, Andrew J Cawthorn, and Wes B Ford. The population biology of abalone (*haliotis* species) in tasmania. i. blacklip abalone (*h. rubra*) from the north coast and islands of bass strait. *Sea Fisheries Division, Technical Report*, 48, 1994.

Alfredo Nazabal, Pablo M. Olmos, Zoubin Ghahramani, and Isabel Valera. Handling incomplete heterogeneous data using vaes, 2018. URL <https://arxiv.org/abs/1807.03653>.

Andrew Y Ng and Michael I Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *NeurIPS*, 2002.

Weili Nie, Brandon Guo, Yujia Huang, Chaowei Xiao, Arash Vahdat, and Anima Anandkumar. Diffusion models for adversarial purification. In *International Conference on Machine Learning (ICML)*, 2022.

- Michiel O Noordewier, Geoffrey G Towell, and Jude W Shavlik. Training knowledge-based neural networks to recognize genes in dna sequences. In *NeurIPS*, 1991.
- Robert Peharz, Steven Lang, Antonio Vergari, Karl Stelzner, Alejandro Molina, Martin Trapp, Guy Van den Broeck, Kristian Kersting, and Zoubin Ghahramani. Einsum networks: Fast and scalable learning of tractable probabilistic circuits. In *International Conference of Machine Learning*, 2020a.
- Robert Peharz, Antonio Vergari, Karl Stelzner, Alejandro Molina, Xiaoting Shao, Martin Trapp, Kristian Kersting, and Zoubin Ghahramani. Random sum-product networks: A simple and effective approach to probabilistic deep learning. In *Uncertainty in Artificial Intelligence*, pages 334–344. PMLR, 2020b.
- Hoifung Poon and Pedro Domingos. Sum-Product Networks: a New Deep Architecture. *UAI 2011*, 2011a.
- Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 689–690. IEEE, 2011b.
- Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features. In *Advances in neural information processing systems*, pages 6638–6648, 2018.
- J. R. Quinlan. Induction of decision trees. *Machine Learning Journal*, 1(1):81–106, 1986.
- J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 1993.
- Tahrima Rahman, Prasanna Kothalkar, and Vibhav Gogate. Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of Chow-Liu trees. In *Machine Learning and Knowledge Discovery in Databases*, volume 8725 of *LNCS*, pages 630–645. Springer, 2014a.



- Tahrima Rahman, Prasanna Kothalkar, and Vibhav Gogate. Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of chow-liu trees. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 630–645. Springer, 2014b.
- Silja Renooij. Same-decision probability: threshold robustness and application to explanation. In *Proceedings of the Ninth International Conference on Probabilistic Graphical Models (PGM)*. PMLR, 2018.
- Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should I trust you?: Explaining the predictions of any classifier. In *SIGKDD*, 2016a.
- Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. In *KDD*, August 2016b.
- Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144. ACM, 2016c.
- Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2018.
- Teemu Roos, Hannes Wettig, Peter Grünwald, Petri Myllymäki, and Henry Tirri. On discriminative bayesian network classifiers and logistic regression. *Machine Learning*, 59(3), Jun 2005. ISSN 1573-0565. doi: 10.1007/s10994-005-0471-6.
- Dan Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1–2):273–302, 1996a.
- Dan Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 1996b.
- Yves Rozenholc, Thoralf Mildenerger, and Ursula Gather. Combining regular and irregular histograms by penalized likelihood. *Computational Statistics & Data Analysis*, 54(12):3313–3323, 2010.

- Donald B Rubin. Inference and missing data. *Biometrika*, 63(3):581–592, 1976.
- Maytal Saar-Tsechansky and Foster Provost. Handling missing values when applying classification models. *Journal of machine learning research*, 8(Jul):1623–1657, 2007.
- Joseph L Schafer. Multiple imputation: a primer. *Statistical methods in medical research*, 1999.
- Peter K. Sharpe and RJ Solly. Dealing with missing values in neural network-based diagnostic systems. *Neural Computing & Applications*, 1995.
- Yujia Shen, Arthur Choi, and Adnan Darwiche. Tractable operations for arithmetic circuits of probabilistic models. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 3936–3944. Curran Associates, Inc., 2016.
- Yujia Shen, Arthur Choi, and Adnan Darwiche. A tractable probabilistic model for subset selection. In *UAI*, 2017.
- Yujia Shen, Arthur Choi, and Adnan Darwiche. Conditional psdds: Modeling and learning with modular knowledge. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- Weijia Shi, Andy Shih, Adnan Darwiche, and Arthur Choi. On tractable representations of binary neural networks, 2020.
- Andy Shih, Arthur Choi, and Adnan Darwiche. A symbolic approach to explaining bayesian network classifiers. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence, IJCAI’18*. AAAI Press, 2018. ISBN 9780999241127.
- Andy Shih, Guy Van den Broeck, Paul Beame, and Antoine Amarilli. Smoothing structured decomposable circuits. In *Advances in Neural Information Processing Systems 32 (NeurIPS)*, 2019.
- Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *ICML*, 2017.

- Dylan Slack, Sophie Hilgard, Emily Jia, Sameer Singh, and Himabindu Lakkaraju. Fooling lime and shap: Adversarial attacks on post hoc explanation methods. In *AAAI/ACM Conference on AI, Ethics, and Society (AIES)*, 2020.
- Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate Kushman. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. *arXiv preprint arXiv:1710.10766*, 2017.
- Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate Kushman. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. In *International Conference on Learning Representations*, 2018. URL <https://openreview.net/forum?id=rJUYGxbCW>.
- Charles Sutton and Andrew McCallum. An introduction to conditional random fields. *Foundations and Trends in Machine Learning*, 2012.
- Srikanth Thudumu, Philip Branch, Jiong Jin, and Jugdutt Jack Singh. A comprehensive survey of anomaly detection techniques for high dimensional big data. *Journal of Big Data*, 7(1):1–30, 2020.
- BETH Twala, MC Jones, and David J Hand. Good methods for coping with missing data in decision trees. *Pattern Recognition Letters*, 29(7):950–956, 2008.
- Ilkay Ulusoy and Christopher M Bishop. Generative versus discriminative methods for object recognition. In *Proceedings of CVPR*, 2005.
- Gerrit JJ van den Burg, Alfredo Nazábal, and Charles Sutton. Wrangling messy csv files by detecting row and type patterns. *Data Mining and Knowledge Discovery*, 33(6):1799–1820, 2019.
- A. Vergari, R. Peharz, N. Di Mauro, A. Molina, K. Kersting, and F. Esposito. Sum-product autoencoding: Encoding and decoding representations using sum-product networks. In *AAAI*, 2018.

- Antonio Vergari, Nicola Di Mauro, and Floriana Esposito. Simplifying, regularizing and strengthening sum-product network structure learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 343–358. Springer, 2015.
- Antonio Vergari, Nicola Di Mauro, and Floriana Esposito. Visualizing and understanding sum-product networks. *preprint arXiv*, 2016. URL <https://arxiv.org/abs/1608.08266>.
- Antonio Vergari, YooJung Choi, Robert Peharz, and Guy Van den Broeck. Probabilistic circuits: Representations, inference, learning and applications. In *Tutorial at the The 34th AAAI Conference on Artificial Intelligence*, 2020.
- Hongzhi Wang, Mohamed Jaward Bah, and Mohamed Hammad. Progress in outlier detection techniques: A survey. *Ieee Access*, 7:107964–108000, 2019.
- David P. Williamson. Bridging continuous and discrete optimization, 2019. URL <https://people.orie.cornell.edu/dpw/orie6334/lecture23.pdf>.
- Jing Xia, Shengyu Zhang, Guolong Cai, Li Li, Qing Pan, Jing Yan, and Gangmin Ning. Adjusted weight voting algorithm for random forests in handling missing values. *Pattern Recognition*, 2017.
- Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017a.
- Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms, 2017b.
- Hongzuo Xu, Yijie Wang, Songlei Jian, Zhenyu Huang, Yongjun Wang, Ning Liu, and Fei Li. Beyond outlier detection: Outlier interpretation by attention-guided triplet deviation network. In *Proceedings of the Web Conference 2021*, pages 1328–1339, 2021.
- LeCun Yann, Cortes Corinna, and Christopher JC Burges. The MNIST database of handwritten digits, 2009.

Shipeng Yu, Balaji Krishnapuram, Romer Rosales, and R Bharat Rao. Active sensing. In *Artificial Intelligence and Statistics*, pages 639–646, 2009.

Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE transactions on neural networks and learning systems*, 30(9):2805–2824, 2019.

Muhammad Bilal Zafar, Isabel Valera, Manuel Gomez Rodriguez, and Krishna P Gummadi. Fairness constraints: Mechanisms for fair classification. *arXiv preprint arXiv:1507.05259*, 2015.

Muhammad Bilal Zafar, Isabel Valera, Manuel Gomez-Rodriguez, Krishna P. Gummadi, and Adrian Weller. From parity to preference-based notions of fairness in classification. In *NeurIPS*, pages 228–238, 2017.